Sistemas Informáticos
Curso 2003 / 04

Sistema de Optimización Distribuido de Redes de Transporte a través de Servicios Web XML

Realizado por:

Juan Carlos Sánchez-Infantes Navas
Carlos Moure García
Juan Mariano de Frutos Bazar

Dirigido por:

Prof. José Jaime Ruz Ortiz
Dpto. Arquitectura de Computadores y Automática

Facultad de Informática
Universidad Complutense de Madrid

INDICE

Autorización a la UCM.

Los autores de este proyecto, Carlos Moure García, Juan Carlos Sánchez-Infantes Navas y Juan Mariano de Frutos Bazar, autorizamos a la Universidad Complutense de Madrid a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la propia memoria como el código, la documentación y/o el prototipo desarrollado.

Fdo. Carlos Moure García

Fdo. Juan Carlos Sánchez-Infantes Navas

Fdo. Juan Mariano de Frutos Bazar

Lista de palabras clave.

1. .NET
2. SERVICIO
3. WEB
4. C#
5. OPTIMIZACIÓN
6. DISTRIBUIDA
7. ALGORITMO
8. GENETICO
9. PLANIFICADOR
10. XML

Resumen

El proyecto es una solución que utiliza la tecnología de los Servicios Web en .NET para la optimización de redes de transporte de forma remota, haciendo uso de otros ordenadores remotos que albergarán los algoritmos de resolución. Para ello, la solución se basa en una arquitectura CLIENTE - SERVIDOR en la que el cliente encapsula los datos topológicos de la red y los envía al servidor que se encarga de optimizarla.

La parte del servidor es una aplicación planificadora que hace uso a su vez de otros Servicios Web para llevar a cabo la optimización de los problemas recibidos. Así pues, este Planificador que recibe los datos a optimizar forma parte de otra arquitectura CLIENTE – SERVIDOR, siendo él el cliente, y como servidores se situarán los Servicios Web de las aplicaciones optimizadoras a las que podrá enviar los datos para su resolución.

Una vez resuelto el problema, el optimizador, a través de su Servicio Web responde con la solución al planificador que enviará, también a través de su Servicio Web, a la aplicación cliente que mandó el problema dicha solución.

El cliente, a su vez, durante el proceso de optimización del problema que mandó, puede consultar el estado en que se encuentra dicho proceso utilizando el Servicio Web que ofrece el planificador para esta tarea. Una vez acabada la optimización el cliente recibe la solución que se guardará en la base de datos que contiene el problema.

This project is a solution that employs Web Service technology developed in .NET to optimize a transport net using resolution algorithms located in remote computers. To accomplish this, the solution is based on a Client-Server architecture in which Client encapsulates the net's topological information and sends it to Server to optimize it.

The side server is a planner application that uses Web Services as well to optimize the received problems. As this planner uses Web Services, it plays the role of client inside another Client-Server architecture. On the server side there will be the optimizer applications that will do the resolution.

Once the problem is solved, the optimizer will send the solution through its Web Service to the planner. The planner will also use its Web Service to send the solution to the client who asked for a solution.

During the optimization process, the client can also check the state process using the Web Service provided by planner for this aim. Once client receives the solution, this will be saved in the database in which the problem is contained

Conceptos Previos

Servicios Web

Básicamente un servicio web es un clase que se publica en un servidor web con soporte para ASP.NET y a cuyos métodos es posible llamar remotamente. Estas clases pueden escribirse en la mayoría de los lenguajes adaptados a .NET, como Visual Basic.NET, Jscript.NET y C#.

Para acceder a un servicio web se pueden utilizar varios protocolos web estándar, como HTTP GET ó HTTP POST, aunque el específicamente diseñado para ello por Microsoft es el *Protocolo de Acceso a Objetos Simple* (Simple Access Protocol o SOAP), que se basa en la utilización del protocolo HTTP para el transporte de mensajes y el lenguaje XML para la escritura del cuerpo de estos mensajes.

Una de las grandes ventajas de los servicios web es que pueden ser accedidos desde cualquier aplicación que sea capaz de generar mensajes e interpretar mensajes escritos en SOAP, aún cuando ésta no esté diseñada para la plataforma .NET. Es más, las aplicaciones que consuman estos servicios no necesitan conocer ni cuál es la plataforma ni cuál es el modelo de objetos ni cuál es el lenguaje utilizado para implementar estos servicios. Otra gran ventaja es que son tremendamente sencillos de escribir, basta hacer unos retoques mínimos al código de una clase cualquiera para convertirla en un servicio Web que podrá ser accedido remotamente; por no mencionar la enorme facilidad y transparencia con la que aquellos clientes del servicio escritos bajo .NET pueden acceder al mismo, gracias a las utilidades generadoras de proxys que se proporcionan.

Microsoft.NET

Microsoft.NET es el conjunto de nuevas tecnologías en las que Microsoft ha estado trabajando durante los últimos años con el objetivo de obtener una plataforma sencilla y potente para distribuir el software en forma de servicios que puedan ser suministrados remotamente y que puedan comunicarse y combinarse unos con otros de manera totalmente independiente de la plataforma, lenguaje de programación y modelo de componentes con los que hayan sido desarrollados. Ésta es la llamada *plataforma .NET*, y a los servicios antes comentados se les denomina *servicios web*.

Para crear aplicaciones para la plataforma .NET, tanto servicios web como aplicaciones tradicionales (aplicaciones de consola, aplicaciones de ventanas, servicios de Windows NT, etc.), Microsoft ha publicado el

denominado kit de desarrollo de software conocido como *.NET Framework SDK*, que incluye las herramientas necesarias tanto para su desarrollo como para su distribución y ejecución y *Visual Studio.NET*, que permite hacer todo lo anterior desde una interfaz visual basada en ventanas.

El concepto de Microsoft.NET también incluye al conjunto de nuevas aplicaciones que Microsoft y terceros han (o están) desarrollando para ser utilizadas en la plataforma .NET. Entre ellas se pueden destacar aplicaciones desarrolladas por Microsoft tales como Windows.NET, Hailstorm, Visual Studio.NET, MSN.NET, Office.NET, y los nuevos servidores para empresas de Microsoft (SQL Server.NET, Exchange.NET, etc.)

C #

El lenguaje C# es el nuevo lenguaje diseñado por Microsoft para su plataforma .NET. En concreto, ha sido diseñado por Scott Wiltamuth y Anders Hejlsberg, éste último también conocido por haber sido el diseñador del lenguaje Turbo Pascal y la herramienta RAD Delphi.

Aunque en realidad es posible escribir código para la plataforma .NET en muchos otros lenguajes, como Visual Basic.NET o JScript.Net, C# es el único que ha sido diseñado específicamente para ser utilizado en esta plataforma, por lo que programarla usando C# es mucho más sencillo e intuitivo que hacerlo con cualquiera de los otros lenguajes. Por esta razón, Microsoft suele referirse a C# como el lenguaje nativo de .NET, y de hecho, gran parte de la librería de clases base de .NET ha sido escrito en este lenguaje.

C# es un lenguaje orientado a objetos sencillo, moderno, amigable, intuitivo y fácilmente legible que ha sido diseñado por Microsoft con el ambicioso objetivo de recoger las mejores características de muchos otros lenguajes, fundamentalmente Visual Basic, Java y C++, y combinarlas en uno sólo en el que se unan la alta productividad y facilidad de aprendizaje de Visual Basic con la potencia de C++.

Quizás el más directo competidor de C# es Java, lenguaje con el que guarda un enorme parecido en su sintaxis y características. En este aspecto, es importante señalar que C# incorpora muchos elementos de los que Java carece (sistema de tipos homogéneo, propiedades, indexadores, tablas multidimensionales, operadores redefinibles , etc.) y que según los benchmarks realizados la velocidad de ejecución del código escrito en C# es ligeramente superior a su respectiva versión en Java.

A continuación se recoge de manera resumida las principales características de C#:

- Dispone de todas las características propias de cualquier lenguaje orientado a objetos: encapsulación, herencia y polimorfismo.

- Ofrece un modelo de programación orientada a objetos homogéneo, en el que todo el código se escribe dentro de clases y todos los tipos de datos, incluso los básicos, son clases que heredan de *System.Object* (por lo que los métodos definidos en ésta son comunes a todos los tipos del lenguaje).
- Permite definir *estructuras*, que son clases un tanto especiales: sus objetos se almacenan en pila, por lo que se trabaja con ellos directamente y no con referencias al montículo, lo que permite accederlos más rápido. Sin embargo, esta mayor eficiencia en sus accesos tiene también sus inconvenientes, fundamentalmente que el tiempo necesario para pasarlas como parámetros a métodos es mayor (hay que copiar su valor completo y no sólo una referencia) y no admiten herencia (aunque sí implementación de interfaces).

- Es un *lenguaje fuertemente tipado*, lo que significa se controla que todas las conversiones entre tipos se realicen de forma compatible, lo que asegura que nunca se acceda fuera del espacio de memoria ocupado por un objeto. Así se evitan frecuentes errores de programación y se consigue que los programas no puedan poner en peligro la integridad de otras aplicaciones.

- Tiene a su disposición un recolector de basura que libera al programador de la tarea de tener que eliminar las referencias a objetos que dejen de ser útiles, encargándose de ello éste y evitándose así que se agote la memoria porque al programador se le olvide liberar objetos inútiles o que se produzcan errores porque el programador libere áreas de memoria ya liberadas y reasignadas.

- Incluye soporte nativo para eventos y delegados. Los delegados son similares a los punteros a funciones de otros lenguajes como C++ aunque más cercanos a la orientación a objetos, y los eventos son mecanismos mediante los cuales los objetos pueden notificar de la ocurrencia de sucesos. Los eventos suelen usarse en combinación con los delegados para el diseño de interfaces gráficas de usuario, con lo que se proporciona al programador un mecanismo cómodo para escribir códigos de respuesta a los diferentes eventos que puedan surgir a lo largo de la ejecución de

la aplicación. (pulsación de un botón, modificación de un texto, etc.).

- Incorpora *propiedades*, que son un mecanismo que permite el acceso controlado a miembros de una clase tal y como si de campos públicos se tratasen. Gracias a ellas se evita la pérdida de legibilidad que en otros lenguajes causa la utilización de métodos *Set()* y *Get()* pero se mantienen todas las ventajas de un acceso controlado por estos proporcionada.

- Permite la definición del significado de los operadores básicos del lenguaje (+, -, *, &, ==, etc.) para nuestros propios tipos de datos, lo que facilita enormemente tanto la legibilidad de las aplicaciones como el esfuerzo necesario para escribirlas. Es más, se puede incluso definir el significado del operador *[]* en cualquier clase, lo que permite acceder a sus objetos tal y como si fuesen tablas. A la definición de éste último operador se le denomina *indizador*, y es especialmente útil a la hora de escribir o trabajar con colecciones de objetos.

- Admite unos elementos llamados *atributos* que no son miembros de las clases sino información sobre éstas que podemos incluir en su declaración. Por ejemplo, indican si un miembro de una clase ha de aparecer en la ventana de propiedades de Visual Studio.NET, cuáles son los valores admitidos para cada miembro en ésta, etc.

XML

El código HTML permite insertar menús, tablas, imágenes o bases de datos en los documentos, pero no permite al usuario que maneje esos elementos como mejor le convenga con la poderosa ayuda del ordenador. Esa es la principal novedad que XML aporta.

Con HTML se pueden hacer accesos a información comparativa en diferentes tiendas por ejemplo, pero nada más. Con XML el usuario podrá ordenar los datos o actualizarlos en tiempo real o realizar un pedido.

La información que manejan las empresas es uno de sus principales activos. Pero lo normal es que esa información esté fragmentada, en diferentes departamentos, ordenadores conectados o no, etc. El reto ahora está en interrelacionar toda esa información para rendir todo su potencial y ponerlo a trabajar para aumentar los beneficios o reducir los costes. Para realizar esto se

necesita un estándar de almacenamiento estructurado que es lo que nos ofrece XML.

XML, es el estándar de Extensible Markup Language. XML no es más que un conjunto de reglas para definir etiquetas semánticas que nos organizan un documento en diferentes partes. XML es un metalenguaje que define la sintaxis utilizada para definir otros lenguajes de etiquetas estructurados.

En primer lugar para entenderlo bien hay que olvidarse un poco, sólo un poco de HTML. En teoría HTML es un subconjunto de XML especializado en presentación de documentos para la web, mientras que XML es un subconjunto de SGML especializado en la gestión de información para la web. En la práctica XML contiene a HTML aunque no en su totalidad. La definición de HTML contenido totalmente dentro de XML y por lo tanto que cumple a rajatabla la especificación SGML es XHTML (Extensible, Hypertext Markup Language).

Los comentarios a partir de los que el compilador generará la documentación han de escribirse en XML, por lo que han de respetar las siguientes reglas comunes a todo documento XML bien formado:

- La información ha de incluirse dentro de etiquetas, que son estructuras de la forma:

  <<etiqueta>> <contenido> </<etiqueta>

  En <etiqueta> se indica cuál es el nombre de la etiqueta a usar.
  Por ejemplo:

  <EtiquetaEjemplo> Esto es una etiqueta de ejemplo </EtiquetaEjemplo>

  Como <contenido> de una etiqueta puede incluirse tanto texto plano (es el caso del ejemplo) como otras etiquetas. Lo que es importante es que toda etiqueta cuyo uso comience dentro de otra también ha de terminar dentro de ella. O sea, no es válido:

  <Etiqueta1> <Etiqueta2> </Etiqueta1></Eetiqueta2>

  Pero lo que sí sería válido es:

  <Etiqueta1> <Etiqueta2> </Etiqueta2></Etiqueta1>

  También es posible mezclar texto y otras etiquetas en <contenido>. Por ejemplo:

  <Etiqueta1> Hola <Etiqueta2> a </Etiqueta2> todos </Etiqueta1>

- XML es un lenguaje sensible a mayúsculas, por lo que si una etiqueta se abre con una cierta capitalización, a la hora de cerrarla habrá que usar exactamente la misma.

- Es posible usar la siguiente sintaxis abreviada para escribir etiquetas sin <contenido>:

  <<*etiqueta***/>**

  Por ejemplo:

  <<EtiquetaSinContenidoDeEjemplo>/>

- En realidad en la <etiqueta> inicial no tiene porqué indicarse sólo un identificador que sirva de nombre para la etiqueta usada, sino que también pueden indicarse *atributos* que permitan configurar su significado. Estos atributos se escriben de la forma <nombreAtributo> **=** **"**<valor>**"** y separados mediante espacios.

  Por ejemplo:

  <EtiquetaConAtributo AtributoEjemplo="valor1" >

  Etiqueta de ejemplo que incluye un atributo

  </EtiquetaConAtributo>

  <EtiquetaSinContenidoYConAtributo AtributoEjemplo="valor2" />

- Sólo puede utilizarse caracteres ASCII, y los caracteres no ASCII (acentos, eñes, ...) o caracteres con algún significado especial en XML han de ser sustituidos por secuencias de escape de la forma *&#<códigoUnicode>*. Para los caracteres más habituales también se han definido las siguientes secuencias de escape especiales:

CARÁCTER	SECUENCIA DE ESCAPE UNICODE	SECUENCIA DE ESCAPE ESPECIAL
<	<	<
>	>	>
&	&	&
‘	'	'
“	"	"

El Tipo de Datos DataSet

El *DataSet* de ADO.NET es una representación de datos residente en memoria que proporciona un modelo de programación relacional coherente independientemente del origen de datos que contiene. Un DataSet representa un conjunto completo de datos, incluyendo las tablas que contienen, ordenan y restringen los datos, así como las relaciones entre las tablas.

Hay varias maneras de trabajar con un DataSet, que se pueden aplicar de forma independiente o conjuntamente. Se puede:

- Crear *DataTables, DataRelations* y *Constraints* dentro de un DataSet, y llenar las tablas con datos mediante programación.
- Llenar el DataSet con tablas de datos desde un origen de datos relacional existente mediante un *DataAdapter*.
- Cargar y hacer persistente el contenido del DataSet mediante XML.

Un DataSet con establecimiento inflexible de tipos también se puede transportar mediante un servicio web XML. El diseño del DataSet hace que sea ideal para transportar datos mediante servicios web XML.

Con ADO.NET es posible llenar un DataSet a partir de una secuencia o un documento XML. Se puede utilizar la secuencia o el documento XML para suministrar datos al DataSet, información de esquema o ambas cosas. La información suministrada desde la secuencia o el documento XML puede combinarse con datos o información de esquema existente ya presente en el DataSet.

ADO.NET también permite crear una representación XML de un DataSet, con o sin su esquema, para transportar el DataSet a través de HTTP con el fin de que lo utilice otra aplicación u otra plataforma compatible con XML. En una representación XML de un DataSet, los datos se escriben en XML y el esquema, si está incluido en línea en la representación, se escribe utilizando el lenguaje de definición de esquemas XML (XSD). XML y el esquema XML proporcionan un formato cómodo para transferir el contenido de un DataSet a y desde clientes remotos.

El DataSet también puede utilizarse junto con datos existentes en un origen de datos cuando se usa con un proveedor de datos de .NET Framework. Éste utiliza un *DataAdapter* para rellenar el DataSet con datos e información de

esquema, así como para resolver cambios relacionados con los datos en el origen de datos.

Algoritmos Genéticos

Introducción a los algoritmos genéticos

Los *algoritmos genéticos* (AG) son métodos sistemáticos para la resolución de problemas de búsqueda y optimización que aplican a estos los mismos métodos de la evolución biológica: selección basada en la población, reproducción sexual y mutación.

Los algoritmos genéticos son métodos de optimización, que tratan de resolver el mismo conjunto de problemas que se ha contemplado anteriormente, es decir, el objetivo es:

hallar ($\mathbf{x}_i,...,\mathbf{x}_n$*) tales que* $\mathbf{F}(\mathbf{x}_i,...,\mathbf{x}_n)$ *sea máximo.*

En un algoritmo genético, tras parametrizar el problema en una serie de variables, $(x_i,...,x_n)$ se codifican en un cromosoma.

Todos los operadores utilizados por un algoritmo genético se aplicarán sobre estos cromosomas, o sobre poblaciones de ellos. En el algoritmo genético va implícito el método para resolver el problema; son sólo parámetros de tal método los que están codificados, a diferencia de otros algoritmos evolutivos como la programación genética.

Hay que tener en cuenta que un algoritmo genético es independiente del problema, lo cual lo hace un algoritmo *robusto*, por ser útil para cualquier problema, pero a la vez *débil*, pues no está especializado en ninguno. Las soluciones codificadas en un cromosoma *compiten* para ver cuál constituye la mejor solución (aunque no necesariamente la mejor de todas las soluciones posibles).

El *ambiente,* constituido por las otras camaradas soluciones, ejercerá una presión selectiva sobre la población, de forma que sólo los mejor adaptados (aquellos que resuelvan mejor el problema) sobrevivan o leguen su material genético a las siguientes generaciones, igual que en la evolución de las especies.

La diversidad genética se introduce mediante mutaciones y reproducción sexual.

En la Naturaleza lo único que hay que optimizar es la supervivencia, y eso significa a su vez maximizar diversos factores y minimizar otros. Un algoritmo genético, sin embargo, se usará habitualmente para optimizar sólo una función, no diversas funciones relacionadas entre sí simultáneamente. Este tipo de optimización, denominada optimización multimodal, también se suele abordar con un algoritmo genético especializado.

Por lo tanto: Un algoritmo genético consiste en hallar de qué parámetros depende el problema, codificarlos en un cromosoma, y se aplican los métodos de la evolución: selección y reproducción sexual con intercambio de información y alteraciones que generan diversidad. En las siguientes secciones se verán cada uno de los aspectos de un algoritmo genético.

Terminología

Algoritmo Genético (Genetic Algorithm) **-** Modelo de aprendizaje computacional que usa una metáfora genético-evolutiva.

Programación Genética (Genetic Programing) - Algoritmos Genéticos aplicados a programas. La Programación Genética es más expresiva que las cadenas de bits de longitud fija de los AGs, aunque los AGs pueden ser más eficientes para algunas clases de problemas.

Gen (Gen) - Partícula de los cromosomas que producen la aparición de caracteres hereditarios.

Genética (Genetic) - Ciencia que estudia los fenómenos relativos a la herencia.

Evolución (Evolution) - Serie de tranformaciones sucesivas que han sufrido los seres vivos desde los tiempos geológicos.

Aptitud (Aptitude) - Medida de la aptitud de un individuo para su supervivencia.

Generación (Generation) - Iteración de la medida de aptitud y la creación de una nueva población por medio de operaciones genéticas.

Reproducción (Reproduction) - Operación genética que origina la creación de una copia exacta de la representación genética de un individuo de la población.

Función de Evaluación (Evaluation Function) - Es la función que elige la próxima generación sobreviviente.

Cruce (Crossover) - Generar dos nuevos hijos a partir de dos cadenas padre.

Mutación (Mutation) - Produce un nuevo hijo a partir de un solo padre con solo cambiar un bit al azar.

Muchedumbre (Crowding) - Fenómeno donde las mejores hipótesis se reproducen siempre y proliferan, reduciendo la diversidad de la población y las posibilidades de evolución.

Orígenes

Los inicios los tenemos situados en la universidad de Michigan, Estados Unidos donde el profesor J. H. Holland introdujo la idea en los años sesenta inicialmente bajo el nombre de Planes Reproductivos, pero se hizo popular en 1975 con el nombre de Algoritmos Genéticos

Prasanna V. Parthasarathy, Martín Pelikan y Arun Ramraj son agregados todos de universidades de Estados Unidos y trabajan en los AG desde 1995, sobre todo enfocados a resolver problemas de optimización mediante AG paralelos.

Los últimos trabajos de Investigación sobre AG los realizan en Illinois Genetic Algorithms Laboratory, donde además podrá encontrar información sobre conferencias y los avances más recientes sobre el tema.

Tendencias Actuales

El éxito actual de los algoritmos genéticos se refleja en conferencias sobre el tema: Illinois Genetic Algorithms Laboratory, una nueva revista internacional dedicada al tema y un sinnúmero de publicaciones alrededor del mundo. Este interés se ha visto ya reflejado en nuestro país, puesto que

centros de investigación como LANIA, Laboratorios Nacionales de Informática Avanzada.

¿Cómo funcionan?

Los algoritmos genéticos consisten en generar una serie de posibles soluciones al azar y en ir creando sucesivas generaciones (soluciones hijas, mezcla de otras soluciones posibles anteriores), dando más importancia y más peso a las mejores soluciones.

El objetivo de los AG es buscar dentro de un espacio de hipótesis candidatas la mejor de ellas. En los AG la mejor hipótesis es aquella que optimiza a una métrica predefinida para el problema dado, es decir, la que más se aproxima a dicho valor numérico una vez evaluada por la función de evaluación.

Ciclo General de un Algoritmo Genético

El Algoritmo Genético estándar se puede expresar en pseudocódigo con el siguiente ciclo:

- Generar aleatoriamente la población inicial de individuos P(0).
- Mientras (numero_generaciones <= máximo_números_generaciones)

```
Hacer:
        {
          Evaluación;
          Selección;
          Reproducción;
          Generación ++;
        }
```

- Mostrar resultados
- Fin de la generación

Función de Evaluación y Selección

La función de evaluación define el criterio para ordenar las hipótesis que potencialmente pueden pasar a formar parte de la siguiente generación.

La selección de las hipótesis que formarán parte de la siguiente generación o que serán usadas para aplicarles los operadores genéticos, puede realizarse de varias formas. Las más usuales son:

- Selección proporcional.
- Selección mediante torneo. Se eligen dos hipótesis al azar.
- Selección por rango. La probabilidad de que una hipótesis sea seleccionada será proporcional a su posición en dicha lista ordenada, en lugar de usar el valor devuelto por la función de evaluación.

Clases de Algoritmos Genéticos

- Algoritmos Genéticos Generacionales.
- Algoritmos Genéticos de estado Fijo.
- Algoritmos Genéticos Paralelos.
- Modelos de Islas.
- Modelo Celular.

Operadores Genéticos

Los dos operadores más comunes son la mutación y el cruce. Existen varios tipos de cruce: cruce simple y cruce doble. La mutación generalmente se aplica después de hacer uso del operador cruce.

Conclusiones

- AG es un método robusto.
- Al comprender los AG la implementación no es complicada y pueden llegar a tener mucha aplicación.
- Los AG son altamente paralelizables.
- Resuelven problemas con un grado de dificultad muy elevado con eficiencia y exactitud.

Introducción al Proyecto

Motivación

El proyecto es un claro ejemplo de un diseño siguiendo una arquitectura CLIENTE-SERVIDOR, y está motivado con el objetivo de optimizar un problema real representado en una base de datos. Este problema va a consistir en la optimización de una red de envío de mercancías. El cliente es miembro de una red de almacenes y demanda una serie de productos para dichos almacenes. Se dispone de una serie de fábricas que suministran dichos productos. El objetivo de la aplicación es mostrar para cada almacén, cuál es la mejor demanda de productos que puede realizar a las diferentes fábricas para que se minimice el coste. Este coste dependerá de la distancia, y otras penalizaciones que se tendrán en cuenta.

La aplicación debe recibir del cliente una base de datos concreta del problema que quiere optimizar, y devolver el resultado de la optimización mediante una estructura de DataSet, y posteriormente generando una base de datos (fichero .mdb). Para realizar la optimización se ha elegido un algoritmo genético. El sistema además del resultado de la optimización, podrá devolver información relativa al algoritmo genético y su ejecución (restricciones, numero de iteraciones, tiempos...). El cliente también dispondrá de un menú de configuración, en el que podrá gestionar de manera muy sencilla las condiciones del algoritmo de optimización genético, como pueden ser: tamaño máximo de la población, número de iteraciones para buscar la solución, número de soluciones requeridas, tipos y tasas de cruces y mutaciones, etc.

Para poder utilizar la aplicación, la máquina cliente necesitará el SGBD Access, y tener definido en una base de datos el problema que desee resolver. Además es necesario poseer una conexión a internet, para poder enviar la información al servidor, y posteriormente, recibir los resultados. En principio, la aplicación cliente consistirá en una aplicación de ventanas.

Descripción General

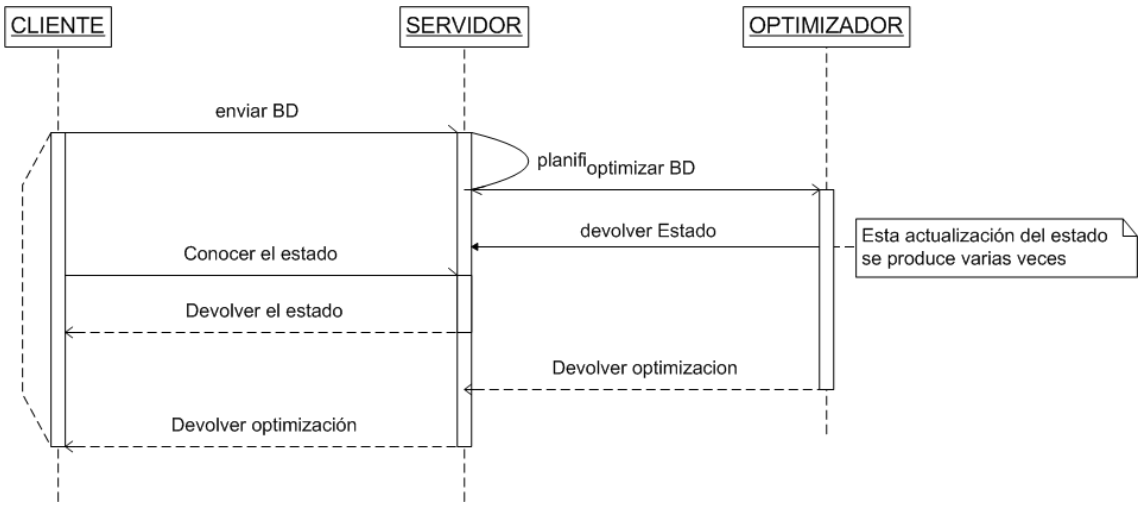
Como ya se ha comentado anteriormente, el proyecto es un claro ejemplo de una arquitectura CLIENTE-SERVIDOR. La aplicación va a estar dividida en tres grandes bloques, como son:

- Aplicación cliente
- Aplicación servidor (planificador).
- Optimizador.

A continuación se comentan los pasos básicos a seguir del funcionamiento de la aplicación:

1. El usuario iniciará la aplicación cliente. El usuario tendrá definida una base de datos mediante tablas, en la que aparecerá la descripción del problema a resolver.
2. El usuario envía la base de datos al servidor, para que éste realice la optimización. Además, puede configurar diferentes parámetros del algoritmo genético.
3. El servidor recibe la base de datos codificada. Realiza la planificación para enviar el problema a uno de los optimizadores, y activa el optimizador que a resultado, que se encargará de ejecutar el algoritmo genético que optimizará la base de datos, y que devolverá los resultados obtenidos.
4. El servidor envía los resultados de la optimización al cliente y dará por finalizada la aplicación.
5. Mientras se está ejecutando la optimización el cliente podrá preguntar por el estado en el que se encuentra la optimización. Como contestación, el servidor le enviará un mensaje, indicándole la situación actual del proceso de optimización.

Se muestra un diagrama de secuencia de la aplicación, que muestra las funcionalidades elementales de la misma.



El funcionamiento típico de la aplicación acarrea muchas incógnitas a tener en cuenta. Entre ellas destacan la posibilidad de incorporar un registro de

usuarios, la estructura que debe seguir la base de datos, el envío de la base de datos a través de internet, la posibilidad de incorporar un planificador y diferentes optimizadores para mejorar la eficiencia de una petición masiva de clientes. Estos temas se describen a continuación.

Se debe discutir la incorporación de un registro de usuarios, que se validará antes de realizar el envío del problema al servidor. Cuando un cliente quiere utilizar la aplicación, se le mostrará una pantalla con el registro de usuarios. Existen dos posibilidades:

- Que se trate de un usuario ya registrado. En su caso, deberá introducir su identificador de usuario y su contraseña.
- Que se trate de un nuevo usuario. Tendrá que registrarse como nuevo usuario, introduciendo un identificador, una clave; y validando dicha clave.

Tras varias versiones de la aplicación al final se ha decidido realizar un registro sin contraseña. El cliente posee su propia base de datos con información referente a él. Un cliente nuevo, no tendrá identificador de cliente, por lo que, antes de solicitar cualquier optimización, deberá registrarse. El servidor le devuelve un identificador en forma de cadena. De esta manera quedará identificador para posteriores peticiones de optimizaciones.

La base de datos donde el usuario describe el problema va a estar formada por cuatro tablas:

- **Suministro**: Representa las fábricas que van a suministrar los productos.

- **Transporte1**: Representa el recorrido de cada producto desde las fábricas hasta el punto intermedio del recorrido o escala.

- **Transporte2**: Representa el recorrido de cada producto desde el punto intermedio hasta los almacenes donde se han destinado dichos productos.

- **Demanda**: Representa las diferentes peticiones que realiza el usuario. Para cada uno de sus almacenes, solicitará un determinado número de unidades de cada producto.

Una vez que el usuario tiene descrito el problema en su base de datos, el cliente debe realizar la conexión CLIENTE-SERVIDOR, y enviarle dicha información. Sin embargo, esta información no puede viajar por la red en forma de tablas. Para ello, se va a utilizar una estructura llamada *DataSet*. El *DataSet*, modela una base de datos, de tal forma que se puede leer como una cadena de caracteres. Posteriormente se explicará como se transforma la base de datos seleccionada por el cliente en la estructura DataSet, que va a ser la información que viaje por la red.

Debido a la duración en la ejecución de un algoritmo genético, y al consumo de procesador que este hace, se ha decidido tener dos algoritmos genéticos distribuidos en sendas máquinas. Por ello, debe haber alguna política de planificación a la hora de asignar un proceso de optimización a un optimizador.

Con todo esto, el sistema va a requerir de los siguientes elementos:

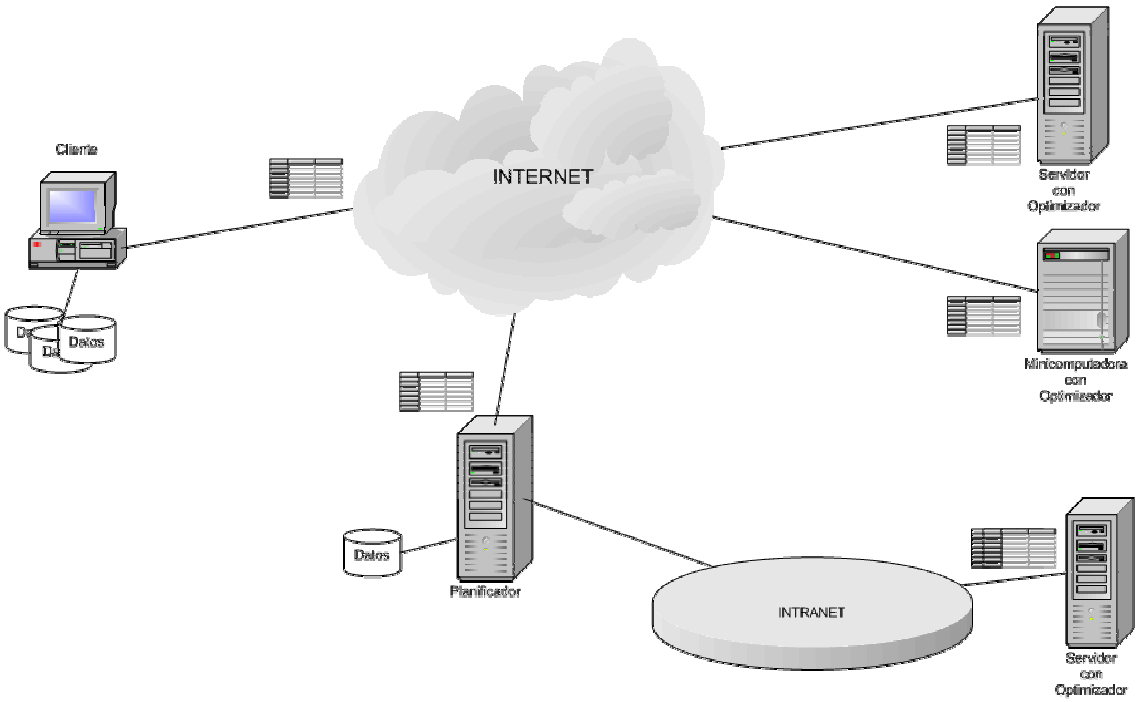
- Una aplicación cliente, que debe leer la base de datos que representa el problema y transformarla a un DataSet para que pueda viajar por la red. A su vez el cliente podrá configurar los diferentes parámetros que identifican a un algoritmo genético. Antes de ser enviada, el cliente debe comprobar que la base de datos descrita es topológicamente correcta, esto es, que debe cumplir un mínimo de restricciones.

- Una aplicación servidor, que recoge el DataSet que le envía el cliente, lo vuelve a transformar en tablas y lo guarda en su base de datos. Además, activa el planificador, que asignará el problema al optimizador adecuado, de forma que se minimice el tiempo de respuesta.

- Varios optimizadores (concretamente 2). En ellos se implementa el algoritmo genético de la aplicación. Lee el problema de la base de datos del servidor y resuelve el problema de optimización, enviando las soluciones al servidor, que a su vez se las reenviará al cliente.

- Una base de datos, donde el servidor depositará el problema que recibe del cliente. Adicionalmente, se dispondrá de otra base de datos con los usuarios registrados.

A continuación se muestra un esquema donde se representan los bloques básicos de que consta la aplicación:



Objetivos

A continuación se muestran los aspectos claves en los que se va a centrar el desarrollo de la aplicación. Estos objetivos se van a dividir en primarios y secundarios.

Objetivos Primarios

Las metas principales de la aplicación es que sea capaz de enviar por la red una base de datos, desde un cliente hasta un servidor; que se optimice el problema que refleja dicha base de datos por medio de un algoritmo genético, y que se devuelvan los resultados a través de la red, mostrándose en la aplicación cliente.

La aplicación cliente, antes de enviar la base de datos, debe comprobar que sea topológicamente correcta, viendo que es posible dar una solución para la especificación del problema dada. También debe ofrecer la posibilidad de configurar los diferentes parámetros que caracterizan un algoritmo genético, a través de un menú.

La aplicación servidor, antes de lanzar el problema al optimizador, debe planificarlo, enviando dicho problema al optimizador que esté libre, o en su defecto, a aquel que se estime que antes va a quedar libre.

Por último, mientras se está produciendo la optimización del problema, el cliente debe puede solicitar el estado actual de la optimización. El servidor deberá proporcionarle un mensaje con dicho estado.

Objetivos Secundarios

En cuanto a las metas secundarias de la aplicación, se considerará el aspecto visual de la misma, ofreciendo al usuario un interfaz sencillo e intuitivo, caracterizado por menús y botones.

También se prestará atención a la posibilidad de llevar a cabo un registro de usuarios, de tal forma que será necesario registrarse para utilizar la aplicación. Con esto, el servidor dispondrá de una base de datos que almacenará a todos los usuarios registrados.

Por último se tendrán en cuenta todas aquellas funciones que ayuden a ampliar las posibilidades de la aplicación, como realizar consultas masivas de los problemas a resolver; configurar el planificador, seleccionando los parámetros que se desean tener en cuenta a la hora de que el servidor seleccione el optimizador que va a encontrar la solución al problema, (diciendo que busque un optimizador libre, uno que tenga poca carga, el más rápido, el más cercano, o que no nos importa el tiempo de respuesta...).

Cliente

Descripción General

El cliente nos facilita una interfaz de comunicación con la aplicación encargada de realizar la optimización de los problemas. Se encargará de la gestión de los procesos (llevar una lista de procesos disponibles, crear nuevos procesos, enviar procesos a optimizar, llevar una lista de procesos enviados a optimizar, ...).

Los procesos los genera el usuario a partir de las Bases de Datos que definen los problemas. Para introducir los datos en las Bases de Datos se deberá utilizar un sistema de Gestión de Bases de Datos, en nuestro caso se utilizará la aplicación Access y el formato de los ficheros será el propio de esta aplicación (con extensión .mdb).

Para que las Base de Datos sean válidas para su uso en nuestra aplicación deben definir el problema a resolver en unas tablas llamadas de una forma determinada que tendrán una estructura determinada y que deberán cumplir una serie de requisitos que verifican que la Red de Transporte definida sea topológicamente correcta.

Una vez creado un proceso se puede enviar para que una aplicación de optimización aplicando un algoritmo de resolución obtenga los valores óptimos para el problema. Para ello el cliente envía el proceso a una aplicación intermedia, llamada planificador, que dispone de la lista de optimizadores disponibles y envía a uno de ellos el problema a resolver.

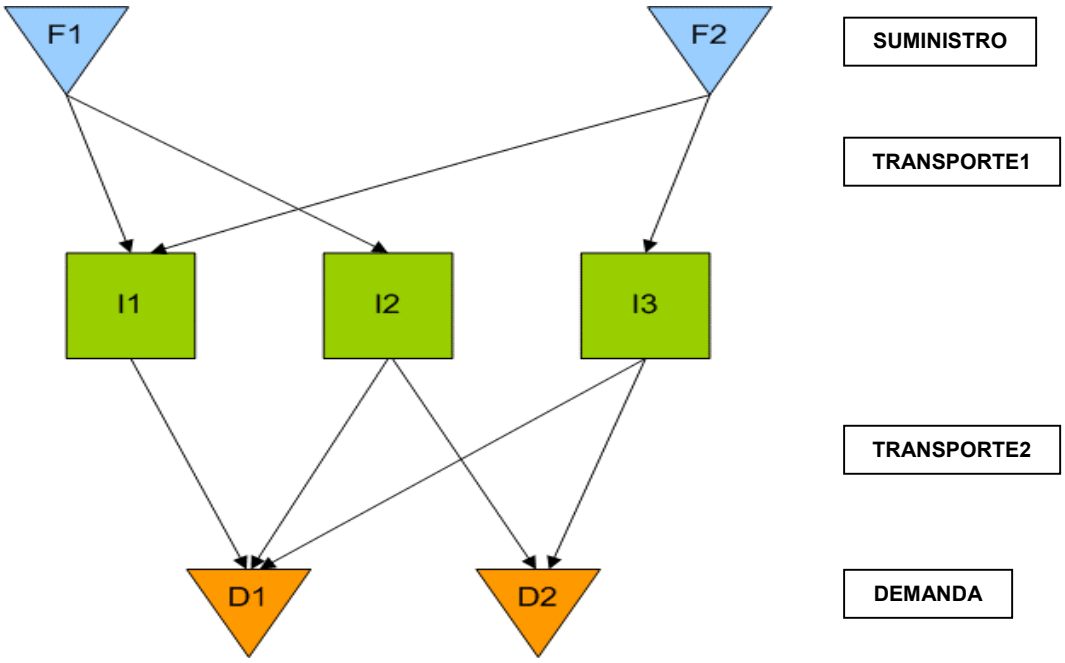
Cuando el optimizador encuentra una solución la comunica al planificador y éste la envía al cliente que mandó el proceso a resolver. En el cliente se recibe y se guarda en la Base de Datos que tiene el problema. Aquí se cierra el ciclo de un proceso, con independencia de que vuelva a mandar otra vez a optimizar.

Para que esta comunicación sea posible de forma que no se cometan errores, cada cliente está identificado de forma única frente al planificador así como los procesos, de tal forma que el planificador sabe en todo momento a qué cliente corresponde cada proceso.

Detalles

Red de Transporte

Es el problema que se quiere resolver. Se representa mediante un grafo en el que están interconectados los diferentes puntos (origen, intermedio y destino), que llevan asociados unas propiedades y valores de coste, por medio de arcos que también llevan asociados unos valores. Estos valores se guardan en las tablas que forman parte de la definición del problema (se explican más adelante). A continuación se muestra una representación gráfica de la red de transportes que se identifica con una base de datos:



En cada origen se fabrican varios productos y de cada producto una cierta cantidad a un coste determinado de fabricación.

En los puntos intermedios se recibe una cierta cantidad de producto desde uno o más orígenes con un coste por unidad determinado de su transporte. Desde estos puntos intermedios se envía una cierta cantidad de cada producto a uno o más destinos con un coste por unidad determinado de transporte.

En cada punto de destino se demanda una determinada cantidad de alguno de los productos que se fabrican en los puntos de origen.

La capacidad de fabricación de cada producto viene determinada por los valores mínimos y máximos de fabricación. En los puntos intermedios también está determinada la capacidad de almacenamiento de cada producto recibido por unos valores mínimos y máximos.

Representación de la Red de Transporte

La red de transporte se define a través de los valores contenidos en las tablas de la base de datos en formato Access. Es importante tener clara la estructura que debe seguir la base de datos donde el usuario describe el problema ya que la aplicación obliga a definirlo a través de dicha estructura. Ésta va a estar formada por cuatro tablas, que representan:

- **Suministro**: Representa las fábricas que van a suministrar los productos. Esta tabla contiene los siguiente atributos:

  Suministro(Fuente,Producto,LimInf,LimSup,Coste).

  - Fuente: Representa a cada una de las fábricas de los productos.
  - Producto: Representa a cada producto fabricado.
  - LimInf: Representa el límite inferior de unidades de ese producto fabricadas por la Fuente.
  - LimSup: Representa el límite superior de unidades de ese producto fabricadas por la Fuente.
  - Coste: Representa el coste fabricación de cada unidad de ese producto por la Fuente.

- **Transporte1**: Representa el recorrido de cada producto desde las fábricas hasta el punto intermedio del recorrido o escala. Esta tabla tiene los siguientes atributos:

  Transporte1(Fuente, Intermedio, Producto, LimInf, LimSup, Coste).

  - Fuente: Es la fábrica de donde viene el producto.
  - Intermedio: Identifica el punto intermedio del recorrido o escala.
  - Producto: Representa el producto que se está transportando por ese recorrido.
  - LimInf: Es el número mínimo de unidades del producto que se pueden transportar por ese camino.
  - LimSup: Es el número máximo de unidades del producto que se pueden transportar por ese camino.
  - Coste: Se trata del coste que ocasiona llevar cada unidad del producto por ese camino desde la Fuente al punto Intermedio.

- **Transporte2**: Representa el recorrido de cada producto desde el punto intermedio hasta los almacenes donde se han destinado dichos productos. Esta tabla tiene los siguientes atributos:

  Transporte2(Intermedio, Destino, Producto, LimInf, LimSup, Coste)

  - Intermedio: Representa el punto intermedio del que parte el producto.
  - Destino: Representa cada una de los almacenes del cliente, a los que llegan los diferentes productos.
  - Producto: Identifica al producto que se está transportando.
  - LimInf: Es el número mínimo de unidades del producto que pueden viajar por ese recorrido.
  - LimSup: Es el número máximo de unidades del producto que pueden viajar por ese recorrido.
  - Coste: Se trata del coste que ocasiona transportar por ese recorrido cada unidad del producto desde el punto Intermedio hasta el Destino.

- **Demanda**: Representa las diferentes peticiones que realiza el usuario. Para cada uno de sus almacenes, solicitará un determinado número de unidades de cada producto. Esta tabla tiene los siguientes atributos:

  Demanda(Destino, Producto, Cantidad)

  - Destino: Representa a los diferentes almacenes de que dispone el cliente.
  - Producto: Representa los productos que el cliente va a solicitar para cada almacén.
  - Cantidad: Es el número de unidades que el cliente requiere de cada producto en cada almacén.

Adicionalmente se ha añadido una nueva tabla, que recoge la configuración que el cliente desea sobre el algoritmo genético. De este modo, el cliente puede modificar los parámetros característicos del algoritmo, y enviarlo junto con el problema para que el optimizador los recoja y los tenga en cuenta. Esta nueva tabla es la siguiente:

- **Config**: Tabla que contiene la configuración que el usuario desea asignar a los parámetros del algoritmo genético. Contiene los siguientes atributos:

  Config(Propiedad, Valor)

  - Propiedad: Es una cadena que indica el parámetro al que se refiere.
  - Valor: Es el valor que tiene el parámetro que se indica en ‘Propiedad’.

Proceso

Objeto que identifica un problema a resolver. Es un objeto compuesto que agrupa varios campos con la información relacionada con un problema a resolver definido en una Base de Datos. Contiene los siguientes campos:

- Una Base de Datos que contiene las tablas que definen la Red de Transporte que se quiere optimizar, así como una tabla con unos valores de configuración utilizados por el Algoritmo de resolución del Optimizador. Esta información en el objeto proceso se guarda en un objeto propio del Framework llamada DataSet.

- Datos identificativos del proceso:
  - Identificador del Cliente que ha generado el proceso (asignado en el Cliente).
  - Identificador del propio proceso (asignado en el Cliente).
  - Nombre de la Base de Datos representada por el DataSet (asignado en el Cliente).
  - Grupo de procesos al que se ha añadido el proceso (asignado en el Cliente).
  - Identificador del Optimizador al que se mandará para su resolución (asignado en el Planificador).

El Cliente se encargará de llevar una lista de procesos generados por el usuario a partir de Bases de Datos que definen problemas que se desean resolver (las Bases de Datos deben estar en formato Access y para que sean válidas deben contener el problema definido en unas tablas llamadas de una forma determinada y cumplir una serie de requisitos que verifican que la Red de Transporte definida sea topológicamente correcta).

¿Qué significa topológicamente correcto?

La red de transporte que se refleja en los valores de las tablas tiene que ser coherente y cumplir unas reglas que se pueden comprobar en el propio cliente antes de mandar la BD a optimizar. Algunas son:

- En la Red (puntos de origen y destino):

$$\sum \text{producto fabricado} \geq \sum \text{producto demandado}$$

- En los puntos intermedios:

$$\sum \text{producto de entrada} \geq \sum \text{producto de salida}$$

- En los puntos de origen e intermedios:

$$\text{Límite inferior} \leq \text{Límite superior}$$

- En los nodos origen para cada producto que se fabrique debe existir un arco que relacione el punto de origen con un punto intermedio para dicho producto

- En cada nodo de destino, la demanda de cada producto debe poder satisfacerse con la suma de las entradas al nodo. Por tanto, debe tener al menos un arco de entrada por cada producto demandado.

- No debe haber ningún arco libre. Esto quiere decir que el origen y destino de un arco deben ser correcto, es decir, debe estar conectado con algún nodo en ambos extremos.

Al recibir la solución en el cliente se podrían realizar otra serie de comprobaciones:

- Generar una tabla en la Base de Datos con la información de las cantidades indicadas en la solución de cada nodo intermedio para saber las cantidades de cada producto que entra y sale y su diferencia.

- Comprobar asimismo que las cantidades indicadas en la solución para los puntos de origen se corresponden con las cantidades demandadas en cada punto de destino para cada producto.

Problema a resolver

El problema consiste en encontrar un conjunto de valores que optimice las restricciones impuestas para satisfacer la demanda de los productos solicitados en los puntos de destino, teniendo en cuenta las limitaciones aplicadas a los puntos de origen y a los intermedios (tanto los referentes a las cantidades como a los costes asociados a cada producto).

Solución al problema

La solución al problema que se recibe en la aplicación cliente desde la aplicación planificador viene en un objeto DataSet. Este objeto contiene tanto las tablas que definen el problema como las de la solución encontrada. Una solución se compone de tres tablas:

- **VarSuministro**: Tabla que define la cantidad de cada producto que se escoge en cada punto de origen.

  VarSuministro(Fuente, Producto, Cantidad)

  - Fuente: Es la fábrica de donde viene el producto.
  - Producto: Representa el producto del que se está seleccionando la cantidad indicada en el siguiente campo.
  - Cantidad: Unidades de producto que se seleccionan del punto de origen indicado en el primer campo. Debe de estar dentro de los límites inferior y superior impuestas por el problema.

- **VarTransporte1**: Tabla que define la cantidad transportada de cada producto desde un punto de origen a uno o más puntos intermedios.

  VarTransporte1(Fuente, Intermedio, Producto, Cantidad)

  - Fuente: Es la fábrica de donde viene el producto.
  - Intermedio: Identificador del punto intermedio al que se transporta el producto indicado en el campo correspondiente.
  - Producto: Representa el producto del que se selecciona, para su transporte, la cantidad indicada en el siguiente campo.
  - Cantidad: Unidades de producto que se seleccionan del punto de origen indicado en el primer campo. Debe de estar dentro de los límites inferior y superior impuestas por el problema.

- **VarTransporte2**: Tabla que define la cantidad transportada de cada producto desde un punto intermedio a uno o más puntos de destino.

  VarTransporte1(Fuente, Intermedio, Producto, Cantidad)

  - Intermedio: Identificador del punto intermedio al que se transporta el producto indicado en el campo correspondiente.
  - Destino: Identificador del punto del que se realiza la demanda del producto indicado en el campo correspondiente.
  - Producto: Representa el producto del que se selecciona, para su transporte, la cantidad indicada en el siguiente campo.
  - Cantidad: Unidades de producto que se seleccionan del punto de origen indicado en el primer campo. Debe de estar dentro de los límites inferior y superior impuestas por el problema.

La Base de Datos del Cliente

Para guardar la configuración de la aplicación cliente y llevar el control de los procesos generados por el usuario, se dispone de una base de datos donde se guarda el estado de las variables que utiliza el cliente. Esta base de datos se encuentra en el mismo directorio que la aplicación.

Tiene dos tablas, que se describen a continuación:

- **AplicConfig**: En esta tabla es donde se guardan los valores de las variables que configuran el comportamiento de la aplicación cliente. La aplicación cliente posee una interfaz para realizar la configuración que actualiza esta tabla una vez realizadas las modificaciones. Contiene los siguientes atributos:

  AplicConfig(Propiedad, Valor)

  - Propiedad: Es una cadena que indica el parámetro al que se refiere.
  - Valor: Es el valor que tiene el parámetro que se indica en 'Propiedad'.

- **Procesos**: Esta tabla contiene la información que define cada proceso que se mandará a optimizar. Al crear un nuevo proceso se guardan los datos correspondientes a éste. Una vez que se mandó el proceso a optimizar se va actualizando el dato que indica el estado en que se encuentra la optimización.

  Procesos(IdProceso, Situacion, Estado, BDAsociada)

  - IdProceso: Nombre que identifica al proceso. Se asigna automáticamente por la aplicación cliente.
  - Situacion: Entero que indica la situación en la que se encuentra el proceso en el cliente. Puede tomar los siguientes valores:
    - 0: Proceso creado.
    - 1: Proceso enviado.
    - 2: Proceso erróneo.
    - 3: Proceso eliminado.
  - Estado: Entero que representa la codificación del estado en el que se encuentra el proceso de optimización. Desde que el cliente crea el proceso de optimización, hasta que se le devuelve la solución, dicho proceso pasa por una serie de estados (creado, enviado, ejecutando, generando solución...). Este campo representa cada uno de los estados.
  - BDAsociada: Es una cadena de caracteres con la dirección de la base de datos que representa el problema de optimización que se quiere resolver. Dicha base de datos estará compuesta por la red de transporte que describe el problema.

Funcionalidad

Funciones disponibles

1. **Crear proceso**

   Da de alta un nuevo proceso asignándole un identificador único en el cliente. Al proceso se le asocia una Base de Datos (en formato Access) que contiene la definición del problema a resolver. El proceso creado se añade a la lista de procesos disponibles y sus datos se guardarán en la tabla “Procesos” de la Base de Datos del Cliente.

2. **Modificar el proceso**

   Modificar datos de configuración del proceso que se usarán en la optimización para configurar el algoritmo de resolución. Además permite cambiar la Base de Datos asociada al proceso. Los datos modificados se actualizan tanto en el objeto proceso de la lista de procesos disponibles como en la tabla “Procesos” de la Base de Datos del Cliente.

3. **Quitar un proceso**

   Elimina un proceso de la lista de procesos disponibles y actualiza el campo “Situación” del registro correspondiente al proceso en la tabla “Procesos” de la Base de Datos del Cliente, con el valor de eliminado, pero no se borra físicamente.

4. **Enviar un proceso a Optimizar**

   Envía un *proceso* al planificador para que éste a su vez lo envíe a un optimizador, éste lo resuelva y responda con una solución, que será remitida por el Planificador al Cliente. Los datos que se envían al planificador son:

   Si se trata de un proceso:
   - Identificador del Cliente que realiza el envío,
   - Identificador del proceso,
   - DataSet con las tablas de la Base de Datos que define el problema,
   - Nombre de dicha Base de Datos.

5. **Configurar el cliente**

   Da la posibilidad de configurar algunas propiedades que influyen en el comportamiento de la aplicación cliente.

Los datos configurables son los siguientes:

IdCliente:

La aplicación Cliente debe poder identificarse frente al Planificador para asociar al Cliente los procesos que envió a optimizar. Para ello el Planificador asignará a cada Cliente que solicite el registro un identificador único. El Planificador llevará la lista de Clientes dados de alta. A la hora de guardar la información relativa a los procesos recibidos incluirá en el registro el identificador del Cliente que se los envió.

La primera vez que se ejecuta la aplicación Cliente no tiene asignado ningún valor a esta propiedad. Se le indicará al usuario mediante un mensaje en pantalla con la posibilidad de realizar el registro en ese momento o hacerlo más tarde.

Como se ha dicho, el identificador del Cliente es necesario para asociar en el Planificador los procesos recibidos con el Cliente que los envió. Así pues, no se podrá realizar desde el Cliente el envío de ningún proceso si el Cliente no está registrado y posee un identificador de Cliente.

Al intentar hacer un envío sin estar registrado, volverá a aparecer en pantalla una nueva ventana para indicarnos esta situación y ofrecernos la posibilidad de realizar el registro en ese mismo momento o postergarlo. Si decidimos no registrar la aplicación no se realizará el envío.

Para llevar a cabo el registro se realizará una llamada al método “addCliente()” del Servicio Web del Planificador. Esta ejecución nos devuelve una cadena de texto con el identificador asignado al Cliente. Este valor se guardará en la propiedad IdCliente y se guardará también en la tabla “Config” de la Base de Datos del Cliente. No se podrá modificar este valor a través de las opciones de la aplicación puesto que es de sólo lectura. Además no es conveniente cambiarlo si ya se han enviado problemas a resolver, pues el identificador se utiliza, tanto en el Planificador como en el Optimizador, para asociar los procesos que pertenecen a cada Cliente.

Validar tológicamente una Base de Datos:

Se puede indicar si se desea realizar las validaciones que comprueban la topología de la Base de Datos que se selecciona al crear o modificar un proceso.

Si la casilla de validación está marcada se realizarán las validaciones. Si se detecta algún error se genera una cadena de texto que indica qué tipo de validación causó el error y se muestra en pantalla para indicárselo al usuario. Si no se marca la casilla no se realiza ninguna validación sobre la Base de Datos.

Este valor también se guarda en la tabla "Config" de la Base de Datos del Cliente.

Para una información detallada sobre las validaciones que se realizan, ver el apartado *"¿Qué significa topológicamente correcto?*" explicado más arriba.

TiempoPing:

Indica el tiempo en segundos que transcurrirán entre las llamadas al Planificador para obtener el estado de los procesos que se enviaron a optimizar y aún están pendientes de resolver. Estos procesos estarán en la lista de procesos enviados.

Internamente el valor de esta propiedad se guarda en la tabla en milisegundos. Por tanto, se realiza la conversión tanto al leer como al guardar el valor de la pantalla de configuración.

StrURL:

Indica la dirección en la que está situado el Servicio Web del Planificador al que se están enviando los procesos a resolver. Esta propiedad guarda su valor en formato URL.

Por ejemplo:

http://147.96.80.155/SIG2/SWPlanificador/SWPlanificador.asmx

La configuración se guarda en la tabla "Config" de la Base de Datos del Cliente. Esta Base de Datos debe estar accesible al arrancar la aplicación. Si no se encuentra, el Cliente no mostrará su Identificador de Cliente (asignado por el Planificador) en el extremo izquierdo de la barra de título de la ventana de la aplicación.

Esta tabla es posible buscarla de forma manual para que la pueda leer el Cliente. Para ello pulsar el botón "Configurar" que se encuentra en la esquina inferior derecha. En ese momento se abrirá la ventana de configuración en la que aparecerán los datos referentes a la configuración del Cliente para poderlos modificar.

En el recuadro con título "Fichero de Configuración" nos aparecerá la dirección de la Base de Datos de configuración del Cliente. Si pulsamos en el botón que tiene los tres puntitos ("...") accederemos a una ventana en la que poder buscar y seleccionar el fichero de configuración.

Sólo hay un valor que no se puede modificar por medio de la aplicación, ya que está bloqueado en esta ventana. Se trata del Identificador del Cliente. Este valor lo asigna el Planificador y no conviene cambiarlo si ya se han enviado problemas a optimizar,

pues este identificador se utiliza, tanto en el Planificador como en el Optimizador, para asociar los procesos que pertenecen a cada cliente.

Si realizamos alguna modificación y aceptamos, los cambios se guardarán en la tabla “Config” de la Base de Datos del Cliente, y además tendrán efecto inmediato en el comportamiento de la aplicación.

Si se cierra la ventana de configuración pulsando el botón “Cancelar”, los cambios efectuados no tendrán ningún efecto y no se grabarán en la Base de Datos de configuración del Cliente.

Restricciones

Las Bases de Datos deben estar en formato Access. Deben tener las tablas apropiadas con la estructura interna apropiada y los nombres apropiados.

La topología de la Red de Transporte debe ser correcta además de cumplir una serie de requisitos.

Mejoras

En esta versión no se ha implementado lo siguiente:

- Listas de Procesos, para enviar varios procesos a la vez a optimizar.
- Asignación dinámica de la referencia del planificador.

Arquitectura del Cliente

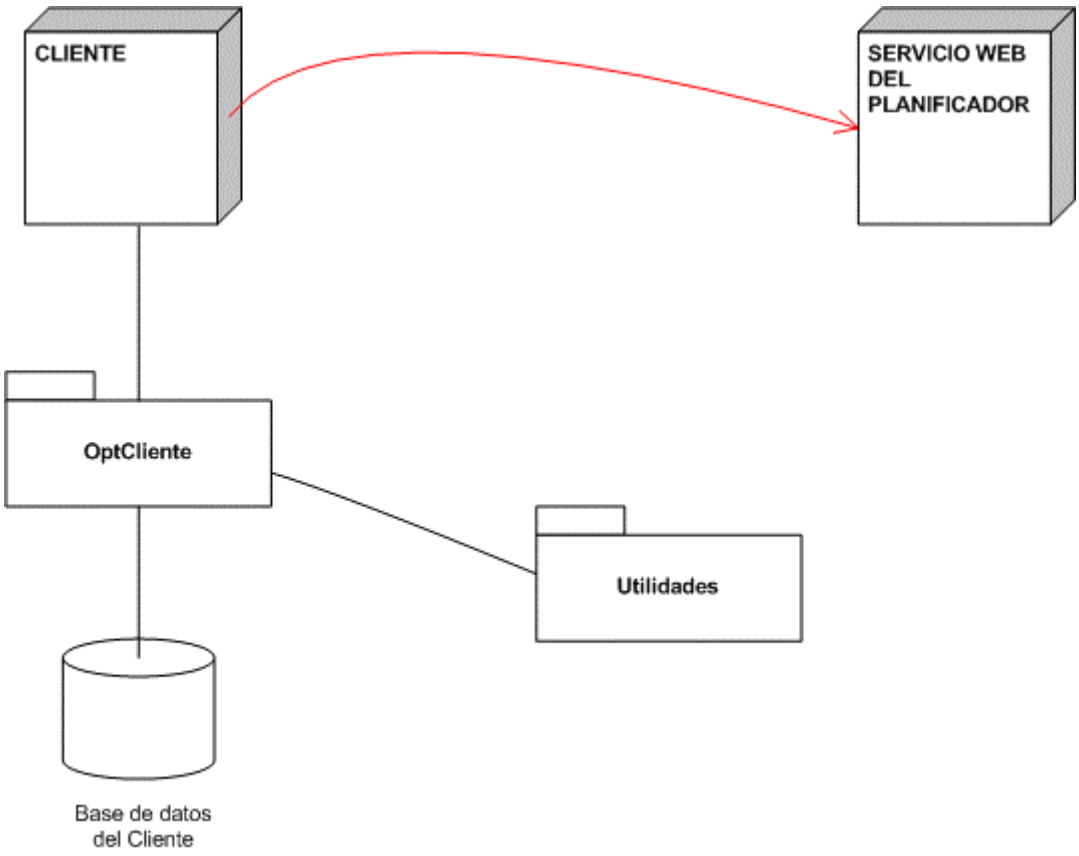
Esquema General y Partes del Cliente

El cliente está compuesto por una interfaz de usuario que implementa un objeto de la librería de clases llamada *OptCliente* que da la funcionalidad del propio cliente. Permite generar procesos que se envían a optimizar para obtener una solución al problema definido. Utiliza para ello el Servicio Web ofrecido por el planificador. Para informar al usuario del estado en que se encuentra la optimización de un proceso el cliente utiliza otro Servicio Web también ofrecido por el planificador.

La librería de clases que se encarga de la funcionalidad del cliente utiliza a su vez otra librería de clases llamada *Utilidades*. Esta contiene la clase *BaseDatos* que proporciona métodos que facilitan consultas sobre una base de datos; cargan bases de datos en una estructura DataSet o guarda el contenido de un DataSet en una base de datos. Se importa dicha librería.

La librería *OptCliente* se apoya en el uso de una base de datos para guardar los valores de la configuración de la aplicación cliente además de llevar la información sobre los procesos creados por el usuario.

A continuación se muestra un esquema de la arquitectura básica del servidor:



La aplicación se apoya en varias clases para dar su funcionalidad. Algunas clases son comunes a todo el proyecto y están recogidas en un espacio de nombres llamado Utilidades.

Clases implementadas

NameSpace OptDistCliente

Se trata de una librería de clases que definen el comportamiento del Cliente del Optimizador Distribuido. Las clases que contiene son:

OptCliente

Opciones

ComprobadorRed

ProcesoEventArgs

Clase OptCliente

Contiene las clases con la lógica de la aplicación cliente, suministrando las estructuras de datos internas, facilitando el acceso a las externas y proporcionando una interfaz sencilla para realizar las llamadas a los servicios web que conforman el conjunto de la solución.

Propiedades

Tiene varias variables privadas que guardan la información necesaria sobre los procesos.

- *private string dirServidor*
  Dirección del Servicio Web del Planificador.
- *private string[] listaBD*
  Lista de las BD disponibles. (aun no se implementa toda su funcionalidad).
- *private Utilidades.ClaseProceso[] listaProcesos*
  Lista de procesos que maneja el Cliente.
- *private int indProceso*
  Indice de la lista de procesos.
- *private string path*
  Directorio de la aplicación.
- *private DataSet DSBD*
  DataSet de la BD que tenemos abierta en ese momento.
- *public string procesoSelec*
  ID del proceso seleccionado para mandarlo al planificador.
- *public Opciones opciones*
  Objeto que contiene la configuración del cliente.
- *private string swURL*
  URL del Servicio Web del Planificador.
- *private string[] listaProcEnv*

lista de procesos enviados al planificador.

- *private int indProcEnv*
  Indice de la lista de procesos enviados.

Eventos para indicar el final de los threads que utilizan los Servicios Web.

- *public delegate void terminaEnvioProceso(object sender, ProcesoEventArgs e)*
  Gestor del evento TerminaEnvioProceso.
- *public event terminaEnvioProceso TerminaEnvioProceso*
  Este evento nos indica que terminó la optimización del proceso que se indica en ProcesoEventArgs.
- *public delegate void terminaGetEstado(object sender, ProcesoEventArgs e)*
  Gestor del evento *TerminaGetEstadoProceso*.

- *public event terminaGetEstado TerminaGetEstadoProceso;*
  Este evento nos indica que terminó la petición del estado del proceso que se indica en ProcesoEventArgs.

Contructores

- public OptCliente(string ID)
  Constructor del cliente con un nombre que lo identifica.
  - ID: Nombre que identificará al cliente

- public OptCliente(string ID, string dir)
  Constructor del cliente con un nombre que lo identifica y la dirección del servidor.
  - ID: Nombre que identificará al cliente.
  - dir: Dirección del Servicio Web del Planificador.

Métodos

- public string[] cargaProcesos()
  Carga la tabla de procesos de la Base de Datos en las listas del cliente creándolos uno a uno. No carga los que está marcados como eliminados. Este método sólo se ejecutara una vez, cuanda se arranca la aplicación cliente.

- public string nuevoProceso(string nombreBD)
  Crea un nuevo objeto proceso con la Base de Datos que se indica por parámetro, la añade la tabla de configuración que utilizará el optimizador si no la tiene, y un identificador para el proceso. A continuación añade el proceso a la lista de procesos y lo guarda el tabla de procesos de la Base de Datos.
  - string nombreBD: Texto que identifica la Base de Datos con la definición del problema.

- private void crearTablaConfig(DataSet ds)
  Crea la tabla de configuración que utilizará el optimizador en el DataSet que se recibe por parámetro. No se comprueba si ya existe la tabla.
    - DataSet ds: Objeto que representa la Base de Datos del proceso que se está tratando.

- public void addProcesoListaProcesos(Utilidades.ClaseProceso p)
  Añade un proceso a la lista de procesos disponibles en el cliente.
    - Utilidades.ClaseProceso p: Objeto con el proceso que se va a incluir en la lista.

- public void addProceso(Utilidades.ClaseProceso p)
  Añade un proceso a la lista de procesos disponibles en el cliente. Además añade un registro a la tabla de procesos de la Base de Datos del cliente.
    - Utilidades.ClaseProceso p: Objeto con el proceso que se va a incluir en la lista.

- public void quitarProceso()
  Elimina un proceso de la lista de procesos disponibles. A su vez quita el proceso de la lista de procesos enviados.

- public Utilidades.ClaseProceso buscarProceso(string nombre)
  Busca el proceso, cuyo nombre se recibe en el parámetro, en la lista de procesos. Devuelve el proceso.
    - string nombre: Nombre del proceso que se va a buscar en la lista.

- public bool buscarProcEnv(string nombre)
  Busca el proceso, cuyo nombre se recibe en el parámetro, en la lista de procesos enviados.
    - string nombre: Nombre del proceso que se va a buscar en la lista.

- public int buscarIndProceso(string nombre)
  Busca el proceso, cuyo nombre se recibe en el parámetro, en la lista de procesos. Devuelve el índice del proceso dentro de la lista de procesos disponibles.
    - string nombre: Nombre del proceso que se va a buscar en la lista.

- public int buscarIndProcEnv(string nombre)
  Busca el proceso, cuyo nombre se recibe en el parámetro, en la lista de procesos enviados. Devuelve el índice del proceso dentro de la lista de procesos enviados.
    - string nombre: Nombre del proceso que se va a buscar en la lista.

- public int numProcesos()
  Indica el número de procesos que hay actualmente en la lista de procesos.

- public int numProcEnv()
  Indica el número de procesos que hay actualmente en la lista de procesos enviados.

- public void addProcEnv(string nombre)
  Añade el nombre del proceso a la lista de procesos enviados al planificador.
  - string nombre: Nombre del proceso que se va a añadir a la lista.

- public void quitaProcEnv(string nombre)
  Elimina un proceso de la lista de procesos enviados.
  - string nombre: Nombre del proceso que se va a quitar de la lista.

- public string[] getListaProcEnv()
  Devuelve la lista de procesos enviados al planificador.

- public bool tablasBD(string nombre, string[] nomTablas)
  Devuelve los nombres de las tablas de la BD. Carga primero la BD en un DataSet. Devuelve los nombres de las tablas en un array.
  - string nombre: Nombre de la base de datos.
  - String[] nomTablas: Lista con los nombre de las tablas de la base de datos.

- public DataTable getTabla(DataSet ds, string t)
  Devuelve una tabla, del Dataset actual, cuyo nombre se pasa por parámetro.
  - DataSet ds: DataSet del que obtener la tabla.
  - string t: Nombre de la tabla a extraer.

- public DataTable getTabla(string t)
  Devuelve la tabla indicada del Dataset.
  - string t: Nombre de la tabla a extraer.

- public void enviarBD()
  Envía la Base de Datos al Servicio Web. Si la BD no está cargada crea el Dataset y carga las tablas de la BD. Este método ya no se utiliza !!!!

- public string registrarCliente()
  Registra el Cliente en la lista de clientes del Planificador, el cual le asigna un identificador de cliente que se utilizará para identificar los procesos que pertenecen al cliente. Este identificador se

guardará en la Base de Datos de configuración y se cargará cada vez que se ejecute el cliente. Deberá se único para cada cliente.

- public void getEstados()
  Realiza la petición del estado en que se encuentra los procesos que estás en la lista de procesos enviados al planificador. Llama al Servicio Web del planificador que da este servicio pasándole el identificador del cliente y el identificador de cada proceso. Recibe un string con la descripción del proceso que va guardando en una lista para actualizar los procesos pendientes. Tener en cuenta que el primer carácter del texto corresponde al índice del estado dentro de la lista de estados posibles del proceso.

- public void getEstado()
  Realiza la petición del estado en que se encuentra el proceso que está seleccionado al planificador. Llama al Servicio Web del planificador que da este servicio pasándole el identificador del cliente y el identificador del proceso. El proceso que se envía es el que se indica en la variable "procesoSelec". Recibe un string con la descripción del proceso. Tener en cuenta que el primer carácter del texto corresponde al índice del estado dentro de la lista de estados posibles del proceso.

- public void actualizaEstados(string[] le, string[] lp)
  Actualiza el estado de los procesos que se reciben en la lista, tanto en la lista de procesos disponibles como en la tabla "Procesos" de la Base de Datos. La lista de estados recibida se corresponde posicionalmente con la lista de procesos enviados.
    - string[] le: Lista con los estados de los procesos.
    - String[] lp: Lista con los nombres de los procesos.

- public void actualizaEstado(string estado, string nombre)
  Actualiza el estado del proceso que se recibe por parámetro, tanto en la lista de procesos disponibles como en la tabla "Procesos" de la Base de Datos.
    - string estado: Valor del estado a asignar.
    - string nombre: Nombre del proceso a actualizar.

- protected virtual void OnTerminaEnvioProceso(ProcesoEventArgs e)
  Provoca la activación del evento que indica que el método ha terminado su ejecución.
    - ProcesoEventArgs e: Nombre del estado que termino. Se encapsula en un objeto de argumentos de evento.

- protected virtual void OnTerminaGetEstadoProceso(ProcesoEventArgs e)
  Provoca la activación del evento que indica que el método ha terminado su ejecución.
    - ProcesoEventArgs e: Nombre del estado que termino. Se encapsula en un objeto de argumentos de evento.

- public void enviarProceso()
  Realiza el envío de un proceso al Planificador. Llama al Servicio Web del planificador que da este servicio pasándole el identificador del cliente, el identificador del proceso, el DataSet que contiene el conjunto de tablas que define el problema y el nombre de la Base de Datos que contiene estas tablas. El proceso que se envía es el que se indica en la variable "procesoSelec":

  - Se busca en la lista de procesos,
  - Se comprueba su validez si procede,
  - Se manda al planificador,
  - Se recoge la respuesta
  - Se guarda en la Base de Datos del proceso.

NameSpace OptDistCliente

Este espacio de nombres engloba varias clase orientadas a la comunicación con el usuario, de las que la más importante es la siguiente.

Clase VistaCliente

Suministra la interfaz de usuario de la aplicación cliente del optimizador distribuido. Utiliza la clase expuesta anteriormente, *OptCliente* para implementar la funcionalidad del cliente.

Propiedades

Tiene varias variables privadas que guardan la información necesaria sobre la configuración.

- private OptDistCliente.OptCliente oc
  Objeto que implementa la funcionalidad de la aplicación cliente.

- private Thread thEnvio
  Hilo de ejecución utilizado para el envío de los procesos al planificador para su optimización.

- private Thread thEstado
  Hilo de ejecución utilizado para la consulta del estado de los procesos que se enviaron a optimizar.

Métodos

- private void abrirBD(object sender, System.EventArgs e)
  Abre una ventana de diálogo para seleccionar el fichero de Base de Datos que contendrá la definición del problema que se asociará a un proceso.

- private void configuraOpciones(object sender, System.EventArgs e)
  Abre el formulario que presenta las opciones de configuración de la aplicación cliente dando la posibilidad de modificarlas y guardar los cambios.

- private void enviarAOptimizar(object sender, System.EventArgs e)
  Envía un proceso a optimizar. Para poder hacer esto el cliente debe estar registrado en el Planificador. El proceso que se envía es el que está seleccionado de la lista de procesos disponible (en el combobox). Se crea un hilo de ejecución (thread) para el envío del proceso, de tal forma que se seguir trabajando con el cliente e incluso enviar otros procesos, para los que también se creará un hilo de ejecución (thread) para cada uno de ellos.

- private void barraEstado(string s, int p)
  Muestra el texto que recibe por parámetro en el panel que se indica también por parámetro en la barra de estado de la aplicación.
    - string s: Texto del mensaje que se quiere mostrar.
    - int p: Identificador del panel donde se mostrará el mensaje.

- private void cambiaProcSelec(object sender, System.EventArgs e)
  Se ejecuta cuando se selecciona un nuevo proceso de la lista de procesos disponibles actualizando los valores de los campos que se muestran en el formulario de datos del proceso. Si no se selecciona ninguno, se resetean los valores presentados.

- private void getEstado()
  Solicita la comprobacion del estado del proceso que está seleccionado en ese momento en el formulario de la aplicación.

- private void getEstados()
  Solicita la comprobacion del estado del proceso que está seleccionado en ese momento en el formulario de la aplicación.

- private void nuevoProceso(object sender, System.EventArgs e)
  Crea un nuevo proceso y lo añade a la lista de procesos.

- private void addProcAListaProc(string nombre)
  Añade un proceso a la lista de procesos del combo que contiene la lista de procesos.
    - string nombre: Nombre del proceso que se quiere añadir a la lista de procesos.

- private void addListaProcAListaProc(string[] nombre)
  Añade una lista de proceso a la lista de procesos del combo que contiene la lista de procesos.
  - string[] nombre: Lista con los nombre de los procesos que se quieren añadir a la lista de procesos.

- private void restauraDatosProceso(object sender, System.EventArgs e)
  Vuelve a poner los valores originales de los datos de configuración del proceso que se han modificado.

- private void limpiaDatosProceso()
  Resetea los valores de los campos del formulario de los datos del proceso.

- private void bloqueaDatosProceso()
  No permite modificar los valores del formulario de datos del proceso.

- private void desbloqueaDatosProceso()
  Permite modificar los valores del formulario de datos del proceso.

- private void rellenaDatosProceso(ClaseProceso p)
  Rellena los controles del formulario de datos del proceso con los valores que contiene el proceso que se recibe por parámetro
  - ClaseProceso p: Objeto que representa el proceso seleccionado.

- private bool ComprobarIDCliente(string s)
  Comprueba si la aplicación cliente está registrada en el planificador a través del identificador de cliente asignado. Si está a blancos es que no está registrado. Si no lo está muestra la opción para registrarse en ese momento. Si es así realiza la llamada correspondiente para hacer el registro del cliente en el planificador.
  - string s: Texto que se mostrará en la ventana informativa que se presenta para avisar al usuario.

- private void ValidaIDCliente(object sender, System.EventArgs e)
  Comprueba si el cliente está registrado en el planificador. Llama a la rutina que hace la comprobación.

- private void modificarDatos(object sender, System.EventArgs e)
  Activa o desactiva los controles del formulario de datos del proceso.

- private void presentaEstado(int estado)
  Decodifica el estado y muestra su descripción en la barra de estado de la aplicación Cliente. Actualiza el valor de la barra de proceso del estado.
  - int estado: Código del estado del proceso seleccionado en pantalla.

- private void TerminaEnvioProceso(object sender, ProcesoEventArgs e)
  Captura el evento *terminaEnvioProceso* y muestra un mensaje al usuario para comunicarle que ha terminado la optimización del proceso indicado.

- private void TerminaGetEstadoProceso(object sender, ProcesoEventArgs e)
  Captura el evento *terminaGetEnvioProceso* y actualiza el estado del proceso indicado.

La librería de clases OptCliente hace uso de otra librería de clases en la que se definen las clases para el acceso a las bases de datos y la encapsulación de los problemas definidos por el usuario para mandarlos al planificador.

NameSpace Utilidades

Se trata de una librería de clases que agrupa las definiciones de clases que se utilizan en el conjunto de la solución. Las clases que contiene son:

BaseDatos

ClaseProceso

Clase BaseDatos

Es una clase que intenta facilitar el manejo de las Bases de Datos desde la aplicación abstrayendo las operaciones de lectura y escritura de éstas y las operaciones de creación de tablas e inserción de datos en dichas tablas.

Propiedades

Posee varias variables privadas que dan información sobre los datos propios de la Base de Datos que referencia:

- *private string bdqnombre*
  Nombre cualificado de la Base de Datos. El nombre cualificado está compuesto del path y del nombre de la Base de Datos.

- *private string bdpath*
  Directorio de la Base de Datos.

- *private string bdnombre*
  Nombre de la Base de Datos.

- *private string bdtipo*
  Tipo de la Base de Datos. Por defecto: Access.

- *private int numtablas*
  Número de tablas que contiene.

- *public OdbcConnection Conexión*
  Objeto conexión de la Base de Datos. Para realizar la conexión se utilizará el acceso de tipo ODBC.

- *private Estado status*
  Objeto Estado de la Base de Datos.

- *private bool evaluada*
  Indica si se mandó alguna vez a optimizar.

- *private bool correcta*
  Indica si la Base de Datos es correcta topológicamente.

Para cada propiedad existe un método set y otro get excepto para la propiedad Conexión.

Contructores

- *public BaseDatos(string path, string nombre)*

  Constructor de la clase BaseDatos con el path y el nombre. Crea el objeto sin conexión.

  - *path*: Nombre del directorio de la nueva Base de Datos.
  - *nombre*: Nombre del fichero de la nueva Base de Datos.

- *public BaseDatos(string nombre)*

  Constructor de la clase BaseDatos con el nombre completo. Crea el objeto sin conexión.

- *nombre*: Nombre del fichero, incluido el path completo, de la nueva Base de Datos.

Métodos

Posee métodos para el manejo de la Base de Datos y de las tablas que contiene.

- public string verConexion()

  Devuelve información sobre la conexión de la Base de Datos.

- public bool nuevaBD(BaseDatos bd)

  Crea un nueva BD copiándola de la BD vacía y con los datos de la BD que se le pasa por parámetro. Devuelve true o false según el éxito de la función.
  Si no existe un fichero con ese nombre se copia la BD vacía y establece la conexión. Si existe no se hace nada y se devuelve false.
  - bd: Objeto que representa una Base de Datos.

- public bool nuevaBD(string nombre)

  Crea un nueva BD copiándola de la BD vacía y con el nombre que se le pasa por parámetro. Devuelve true o false según el éxito de la función.
  Si no existe un fichero con ese nombre se copia la BD vacía y establece la conexión. Si existe no se hace nada y se devuelve false.
  - nombre: Directorio y Nombre con extensión del fichero de la nueva BD.

- public bool nuevaBD(string path, string nombre)

  Crea un nueva BD copiándola de la BD vacía y con el nombre que se le pasa por parámetro. Devuelve true o false según el éxito de la función.
  Si no existe un fichero con ese nombre se copia la BD vacía y establece la conexión. Si existe no se hace nada y se devuelve false.
  - path: Nombre del directorio de la nueva BD.
  - nombre: Nombre del fichero de la nueva BD.

- public OdbcConnection obtenerConexion(string BD)

  Crea una conexión para la Base de Datos, pero no la abre. Devuelve un objeto OdbcConnection con la conexión a la Base de Datos.

  Cadena de conexión:
  "Driver={Microsoft Access Driver (*.mdb)};Dbq="+BD+";Uid=;Pwd="

  - BD: Nombre completo de la Base de datos de la que obtener una conexión.

- public void abrirConexion(string BD)

  Si la conexión está cerrada la abre. Si no tiene conexión la obtiene previamente.

- public bool cargar(DataSet DS)

  Lee la BD y carga sus tablas en el DataSet indicado. Crea un array de adaptadores con un adaptador por cada una de las tablas de la Base de Datos. Devuelve true o false según el éxito de la función. Si no tiene conexión, la obtiene aquí. Si la BD no está abierta, la abre. Una vez realizada la carga en el DataSet se cierra la conexión.
  - DS: DataSet donde cargar las tablas de la BD.

- public DataSet cargarDS()

  Lee la BD y carga sus tablas en el DataSet indicado. Crea un array de adaptadores con un adaptador por cada una de la tablas de la Base de Datos. Devuelve el DataSet con la BD cargada.

- public int ejecutaConsulta(DataSet ds, string tabla, string cmd)

  Ejecuta la consulta indicada en el parámetro "cmd" sobre la BD y guarda el resultado en la tabla "tabla" en el Dataset "ds" indicado. Devuelve el número de filas de la tabla insertada.
  - ds: DataSet en el que se guardan el resultado de la consulta.
  - tabla: Nombre de la tabla en la que se guarda el resultado en el DataSet.
  - cmd: Cadena de texto especificando la consulta a realizar.

- public bool ejecutaConsultas(DataSet ds, string[] tablas, string[] cmds)

  Ejecuta varias consultas indicadas en el parámetro array "cmds" sobre la BD y guarda el resultado en las tablas indicadas en el parámetro array "tablas" en el Dataset "ds" indicado.
  - ds: DataSet en el que se guardan los resultados de las consultas.
  - tablas: Nombres de la tabla en las que se guarda los resultados en el DataSet.
  - cmds">Cadenas de texto especificando las consultas a realizar.

- public bool ejecutaActualizacion(string tabla, string campo, string valor, string cond)

  Ejecuta la actualización de un registro de una tabla de la Base de Datos.
  - tabla: Nombre de la tabla que se quiere actualizar.
  - campo: Nombre del campo al que se asignará el nuevo valor.
  - valor: Nuevo valor que se asignará al campo.
  - cond: Condición SQL para seleccionar el registro.

- public int altaRegs (DataTable dt, OdbcConnection cnn)

  Da de alta los registros de la tabla indicada en la BD indicada por la conexión. Devuelve el número de filas afectadas en la operación, o (-1) si algo va mal. Construye la lista de campos y valores y llama al método altaReg.
  - dt: DataTable con los registros que se van a dar de alta en la BD.
  - cnn: Objeto que representa la conexión con la BD.

- public bool altaReg (string tabla, string campos, string valores, OdbcConnection cnn)

  Da de alta un registro en la tabla indicada, con los campos indicados, los valores indicados y en la BD indicada por la conexión. Devuelve true o false según el éxito de la función.
  - tabla: Nombre de la tabla de la BD.
  - campos: Lista de campos del registro separados por comas.
  - valores: Lista de valores de los campos del registro separados por comas.
  - cnn :Objeto que representa la conexión con la BD.

- public int altaTablas (DataSet ds, OdbcConnection cnn, bool cargar)

  Da de alta en la BD las tablas nuevas del DataSet y carga sus datos. Devuelve el número de tablas afectadas en la operación, o (-1) si algo va mal. Para cada tabla del DataSet se comprueba si existe en la BD. Si no existe se da de alta llamando al método altaTabla y se cargan sus registros, si se indica en el parámetro.
  - ds: DataSet que contiene las tablas.
  - cnn: Objeto que representa la conexión con la BD.
  - cargar: Indica si se cargan los registros de las tablas.

- public bool altaTabla (DataTable dt, OdbcConnection cnn)

  Da de alta en la BD la tabla que se especifica en los parámetros. Actualiza la propiedad que indica el número de tablas de la Base de Datos. Devuelve true o false según el éxito de la función. Para cada columna de la tabla se obtiene su nombre y su tipo construyendo una cadena que utiliza el comando CREATE TABLE.
  - dt: DataTable que se va a dar de alta en la BD.
  - cnn: Objeto que representa la conexión con la BD.

- public bool guardaBD(string bd, DataSet ds)

  Guarda el contenido de un DataSet en otra BD. Si no existe, se crea la BD. Devuelve true o false según el éxito de la función. Si la BD especificada es la misma que la referenciada por este objeto llamamos al otro método con el DataSet.

  Creamos un nuevo objeto BaseDatos para la nueva Base de Datos. Si la BD no existe la creamos nueva vacía. Si no tiene una conexión, la obtiene aquí. Si la BD no está abierta, la abre. Da de alta las tablas en la Base de Datos y carga los datos.

  - bd: Nombre de la BD con el path completo.
  - ds: DataSet con las tablas que se grabarán en la BD.

- public bool guardaBD(DataSet ds)

  Guarda el contenido del DataSet indicado, en la propia BD. Devuelve true o false según el éxito de la función.

  Si no tiene una conexión, la obtiene aquí. Si la BD no está abierta, la abre. Da de alta las tablas en la Base de Datos y carga los datos.

  - ds: DataSet con las tablas que se grabarán en la BD.

- public string [] getNomTablas()

  Devuelve los nombres de las tablas de la Base de Datos. Utiliza una conexión OleDB para obtener el esquema de la Base de Datos. Devuelve un array con los nombres de las tablas de la Base de Datos.

  Para obtener las tablas de la Base de Datos no nos sirve ODBC, porque no tiene utilidades para esto. Usaremos una conexión OleDb para obtener el esquema de la Base de Datos (tenemos que establecer las restricciones de extracción de los datos del esquema con el siguiente array: object [] restricciones = {null, null, null, "TABLE"}) y a partir de éste sacamos los nombres de las tablas y su número (que devolvemos en el parámetro 'n').

  Cadena de conexión:

  “Provider=Microsoft.Jet.OLEDB.4.0; Data Source=" +QNombre

  El esquema se obtiene en un objeto DataTable con una fila para cada tabla que contenga la Base de Datos. Después miramos qué columna es la que contiene los nombres de las tablas y para cada fila del DataTable leemos el nombre de la tabla.

Clase ClaseProceso

Representa un proceso de optimización que se va a enviar a uno de los dos optimizadores de que consta la aplicación. Contiene el identificador del cliente al que pertenece el proceso, así como su propio identificador dentro de ese cliente. Tiene información sobre la base de datos a la que se asocia el proceso, así como el optimizador donde se va a llevar a cabo la optimización. Por último posee el DataSet; que en definitiva es la información que se le manda al optimizador.

Propiedades

- private string idCliente
  Identificador del Cliente que lo ha generado. Asignado por la aplicación Cliente.

- private string idProceso
  Identificador del propio proceso. Asignado por la aplicación Cliente.

- private string bdAsociada
  Nombre de la Base de Datos que se asoció al crear el proceso. Asignado por la aplicación Cliente.

- private int optAsociado
  Identificador del Optimizador al que el planificador lo envió para optimizar. Este dato lo asigna la aplicación Planificador.

- private string status
  Indica el estado en el que se encuentra el proceso. puede pasar por varios estados desde que se crea en la aplicación Cliente , se manda a la aplicación Planificador y se resuelve en un Optimizador.

- private DataSet ds
  Objeto que representa los datos del problema a resolver leídos de la Base de Datos que tiene asociada el proceso. Estos datos viajan a la aplicación Planificador.

Constructores

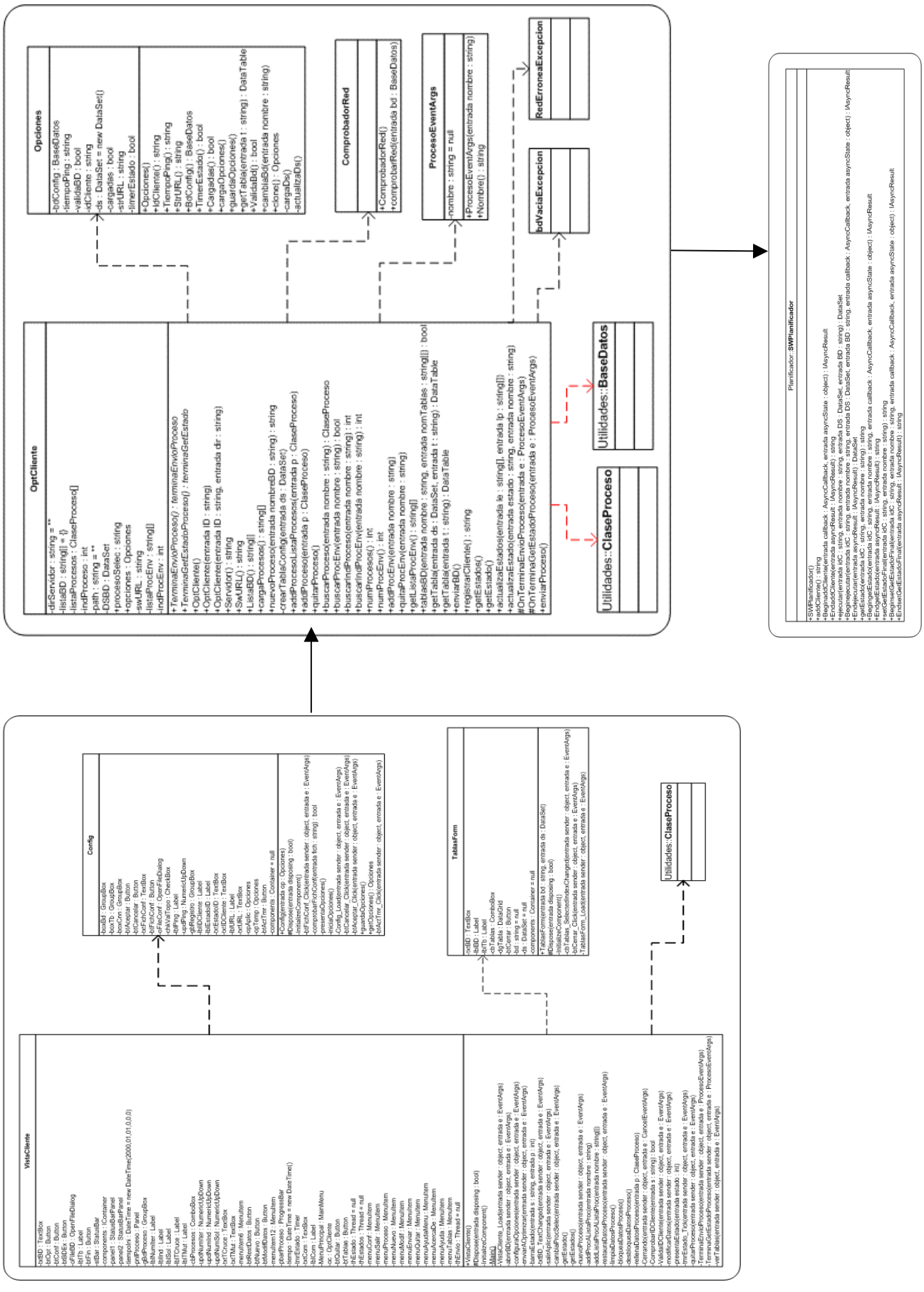
- public ClaseProceso(string idC)
  Constructor con un parámetro. Crea un nuevo proceso con el identificador del cliente.
  - idC: Identificador del cliente

- public ClaseProceso(string idC, string idP)
  Constructor con dos parámetros. Crea un nuevo proceso con el identificador del cliente y con el identificador del proceso.
  - idC: Identificador del cliente
  - idP: Identificador del proceso

Para cada propiedad existe un método set y otro get excepto para la propiedad Conexión.

Diagrama de Clases UML del Cliente

En la página siguiente se especifica en detalle un diagrama de clases de la aplicación cliente. En él se pueden ver las diferentes clases que intervienen en este módulo, sus atributos y sus métodos; así como las relaciones entre ellas:


OptCliente
Opciones
ComprobadorRed
ProcesoEventArgs
bdVaciaExcepcion
RedErroneaExcepcion
Utilidades::BaseDatos
Utilidades::ClaseProceso
Planificador::SWPlanificador
VistaCliente
Config
TablasForm
Utilidades::ClaseProceso

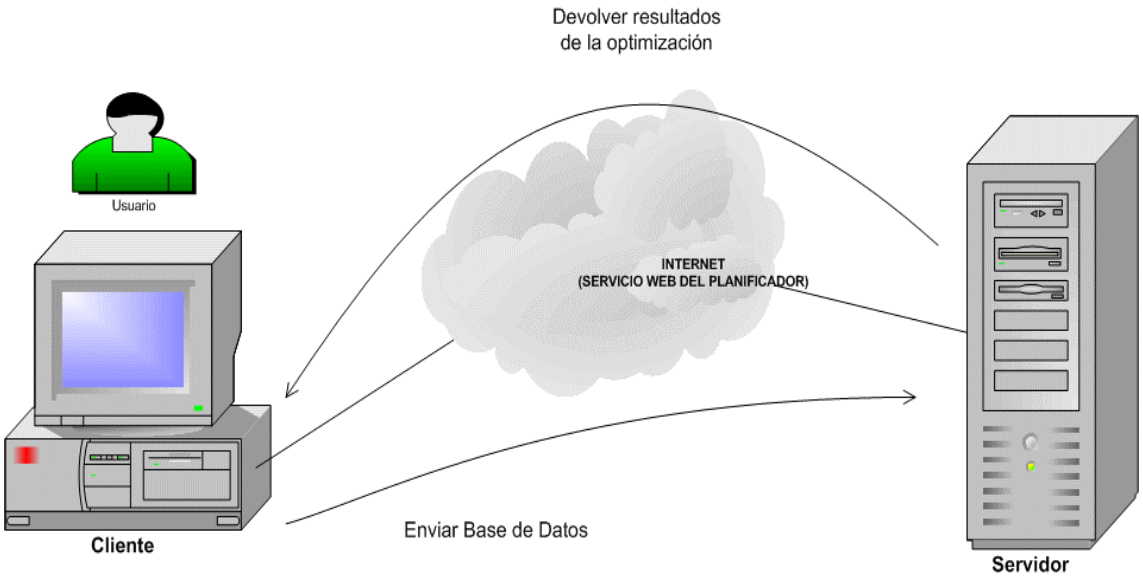
Planificador

Descripción General

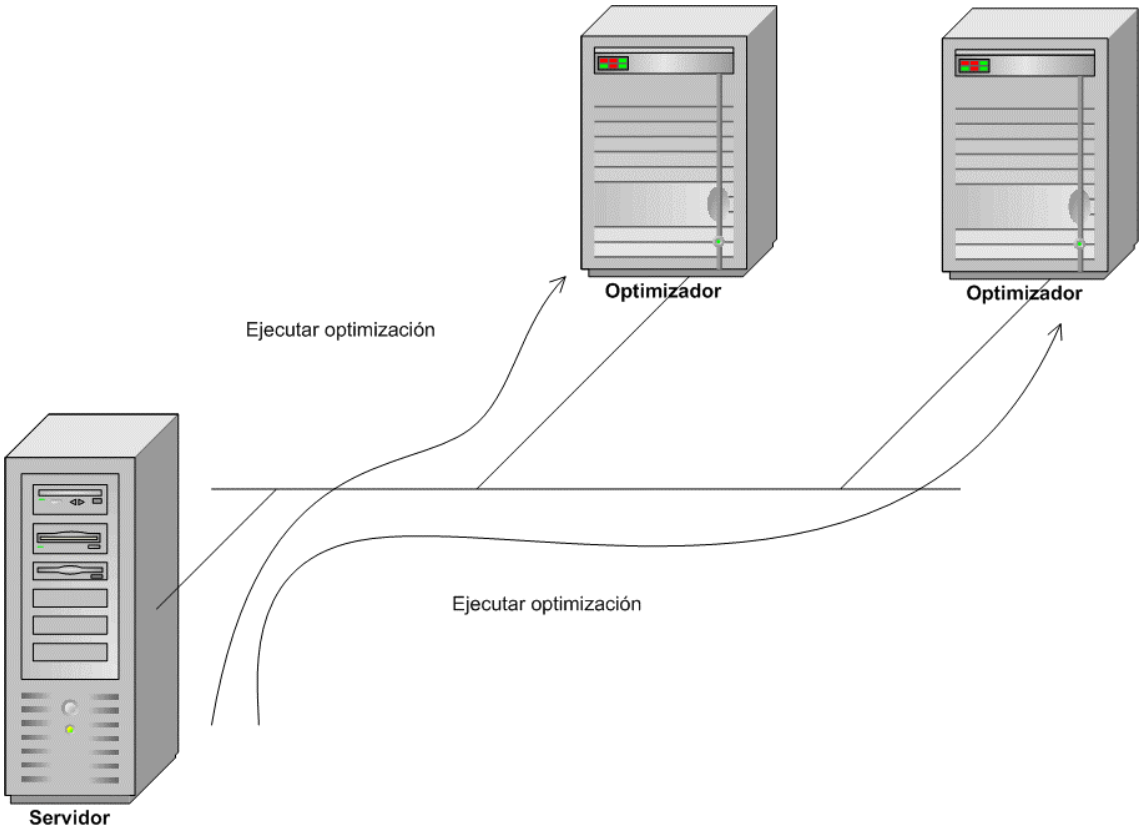
El planificador es un módulo que enlaza la aplicación cliente con los optimizadores. Recibe de éste una serie de problemas de optimización a resolver (procesos), los planifica y los envía al optimizador correspondiente. Una vez que el optimizador ha encontrado una solución, el planificador, tras recibirla, se la reenvía al cliente.

La comunicación que existe entre el planificador y las aplicaciones cliente y optimizador así como el cambio de información entre ellas se realiza a través de servicios web. Este enlace tiene lugar a través de Internet entre el cliente y el servidor, mientras que para la comunicación entre el servidor y los optimizadores se produce mediante una red local. Incluso puede que en un mismo procesador se sitúen el planificador y uno de los optimizadores a utilizar.

Esquema de la comunicación entre el cliente y el servidor:



Esquema de la comunicación entre el servidor y los optimizadores (una posibilidad). En este dibujo, la comunicación entre el servidor y los optimizadores es mediante una red local. El servidor y los optimizadores están en máquinas distintas:



El DataSet es la estructura de datos que lleva la información del problema a optimizar, así como las restricciones que se quieren dar sobre el algoritmo de optimización; y es la información que viaja por la red, junto con otra información adicional encargada de la coherencia en las comunicaciones.

Además de recibir, planificar y reenviar los problemas de optimización que le llegan, el planificador debe realizar una gestión sobre los clientes que van a hacer uso de la aplicación, así como de los procesos de optimización que van a enviar. De esta manera, el planificador necesitará guardar información sobre cada cliente, sus procesos, el estado en el que se encuentra cada uno de ellos, la base de datos a la que hace referencia el problema... Por otro lado, también deberá realizar una gestión de los optimizadores respecto a cuál se llamó por última vez, y si se encuentran o no optimizando.

Detalles

El Problema de la Conservación del Estado

En las primeras versiones de la aplicación, se realizó el planificador sin utilizar un servicio web para comunicar el cliente con el servidor. Dicho planificador poseía variables privadas dentro de una clase que identificaban a los clientes, sus procesos de optimización, la información que codifica el problema a resolver, y el estado de los optimizadores para llevar a cabo la planificación. La actualización de las variables se realizaba de forma coherente de una llamada a otra al planificador.

Posteriormente, se realizó una nueva versión en la que se utilizaba un servicio web. El cliente lo utilizaba enviando la información necesaria. El servicio web, realizando las llamadas pertinentes, le suministraba al cliente la solución al problema de optimización planteado.

Tras realizar varias pruebas con esta versión, se observó que el servicio web no mantenía el estado de sus variables, trabajando siempre con sus valores por defecto. Por esta razón el grupo de trabajo se vio obligado a mantener el estado de las variables del planificador mediante una base de datos.

Por este motivo, cada vez que se llame al planificador para consultar un dato, éste deberá leerse de la base de datos. Del mismo modo, si se modifica un dato del planificador, o se da de alta un nuevo cliente, o se añade un nuevo proceso para optimizar, esta información deberá guardarse en la base de datos del planificador.

La Base de Datos del Planificador

Con motivo de los problemas que se describen en el apartado anterior, el planificador posee una base de datos donde se guarda el estado de las variables que utiliza el servidor. Estas variables hacen referencia a los clientes, los procesos de optimizador que manda cada cliente, y el estado de los optimizadores para realizar la planificación.

La base de datos del planificador se encuentra en el directorio principal del servicio web del planificador.

Esta base de datos posee tres tablas, que se describen a continuación:

- **Clientes**: Se trata de una tabla donde se encuentran registrados los clientes que han utilizado la aplicación. Contiene los siguientes atributos:

  Clientes(idCliente, NombreCliente)

  - IdCliente: Es un entero que identifica al cliente.
  - NombreCliente: Es una cadena de caracteres que posee el nombre del cliente. Este nombre se le devuelve a la aplicación cliente y sirve como clave para acceder a esta tabla.

- **Procesos**: Esta tabla guarda la información que se necesita sobre cada proceso de optimización. El cliente envía una serie de datos identificativos junto con el DataSet del problema a optimizar, y esos datos se guardan en esta tabla para posteriormente realizar consultas sobre el estado de cada proceso de optimización de un cliente determinado.

  Procesos(idCliente, idProceso, nombreProceso, EstadoProceso, OptAsociado, BDAsociada)

  - IdCliente: Entero que identifica al cliente. Se obtiene del identificador de la tabla *Clientes*.
  - IdProceso: Entero que identifica al proceso. Representa el ordinal del número de procesos que ha enviado un cliente determindado.
  - NombreProceso: Cadena de caracteres que identifica al proceso. Este dato lo proporciona el cliente, ya que es él quien asigna nombre a los procesos de optimización.
  - EstadoProceso: Es un entero que representa la codificación del estado en el que se encuentra el proceso de optimización. Desde que el cliente crea el proceso de optimización, hasta que se le devuelve la solución, dicho proceso pasa por una serie de estados (creado, enviado, ejecutando, generando solución...). Este campo representa cada uno de los estados.
  - OptAsociado: Es un entero que representa el optimizador que va a resolver el problema de optimización. Una vez que el servidor ha realizado la planificación, se conoce cuál va a ser este optimizador. En ese momento se actualiza este campo.
  - BDAsociada: Es una cadena de caracteres con la dirección de la base de datos que representa el problema de optimización que se quiere resolver. Dicha base de datos estará compuesta por la red de transporte que describe el problema.

- **Planificador**: Esta tabla contiene los valores de las variables que el servidor necesita para realizar la planificación. Se compone de un único registro.

  Planificador(ultimoProceso, ActivoOpt1, ActivoOpt2)

  - UltimoProceso: Es un entero que almacena el identificador del último optimizador al que se llamó.
  - ActivoOpt1: Es un entero que indica si el optimizador 1 está activo o no, es decir, si está optimizando.
  - ActivoOpt2: Es un entero que realiza las mismas funciones que el parámetro anterior, pero para el optimizador 2.

La Planificación

Se sabe que un algoritmo genético que proporcione una buena solución al problema que se le plantea tiene un tiempo de respuesta relativamente elevado, dependiendo de la magnitud de la información que debe procesar. Para esta aplicación, un tiempo medio aceptable puede situarse alrededor de los 30 minutos.

El uso de servicios web para la optimización parece que evita este problema, ya que se pueden realizar varias peticiones al mismo servicio, que se atienden de manera independiente. Sin embargo, también se conoce que los algoritmos genéticos son densos en uso del procesador; con lo que un único servicio web posiblemente quedaría saturado con unas pocas peticiones de optimización. Parece claro que una alternativa buena para distribuir el trabajo es hacer uso de varios optimizadores, cada uno en una máquina distinta.

Es muy posible que un usuario quiera mandar un mismo problema de optimización varias veces, cambiando algunos parámetros y/o variables, para conocer cuál es la mejor solución al problema. Por tanto parece necesario tener un algoritmo de planificación que envíe los problemas de optimización a los diferentes optimizadores de una manera razonable para evitar aumentar de forma intolerable el tiempo de espera y para conseguir una mayor carga de trabajo, evitando la inactividad de uno de los procesadores mientras que el otro se ve saturado. Esta necesidad se acentúa aún más si se considera la aplicación mediante una arquitectura Cliente-Servido vía Internet, donde puede haber varios usuarios (clientes) solicitando peticiones de optimización.

Por estas razones se ha diseñado una aplicación intermedia entre el cliente y el optimizador. El servidor, a parte de ofrecerle al cliente los servicios internos de la aplicación como puede ser la consulta del estado de un problema

de optimización, realiza una planificación de los procesos de optimización que recibe.

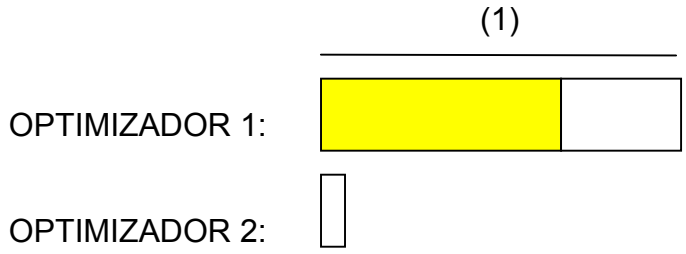
Sobre los optimizadores, comentar las siguientes restricciones referentes a la planificación:

- Un proceso de optimización se le asigna a un único optimizador. Es decir, si se decide que el optimizador 1 va a realizar la optimización de un determinado proceso, él es el único que la realiza, sin posibilidad de que el trabajo se reparta entre los optimizadores.
- El optimizador no abandona el proceso de optimización hasta que no haya terminado y devuelto la solución. Es decir, un optimizador no deja bloqueado y sin terminar una optimización para atender a otra, sino que las va atendiendo conforme va terminando.

En las primeras versiones del planificador se optó por realizar una política de turno rotatorio. Esto es, se van alternando los optimizadores a cada petición de optimización que llega. Por ejemplo, suponer la siguiente situación:

"Se encuentran los dos optimizadores inactivos, y llega un proceso para optimizar. El planificador asigna la optimización al optimizador 1. Llega una segunda petición, y el planificador se la asigna al optimizador 2. Posteriormente llega una tercera, asignándosela nuevamente al optimizador 1. Así se van alternando la ejecución de optimización de los optimizadores".

Sin embargo esta política planteaba un problema: ¿Qué sucede si un optimizador ya ha terminado, y la nueva petición de optimización se la debemos asignar al otro?. Suponer la siguiente situación:



(1). Indica el tiempo que tarda en ejecutarse el algoritmo del optimizador 1

"Amarillo": Indica el tiempo transcurrido.

"Blanco": Indica el tiempo que falta.

La situación es que el optimizador 1 aún no ha terminado de realizar la optimización, mientras que el optimizador 2 sí, y está sin hacer nada. Es posible que la planificación decida que el siguiente problema se debe enviar al optimizador 1, a pesar de que podría realizarla el optimizador 2, ya que ha terminado. Claramente, con la política planteada no se aprovecha al máximo los recursos del sistema, por tanto, habría que mejorarla.

En las siguientes versiones del planificador se plantea la posibilidad de enviar dos optimizaciones seguidas al mismo optimizador. Sigue siendo una política de turno rotatorio, pero con ciertas mejoras.

A continuación se detallan las fases de la planificación:

- Si ambos optimizadores están inactivos, el proceso de optimización se envía al optimizador 1.

- Si la anterior optimización se envió al optimizador 1, se debe ver si éste ha terminado.
    - Si el optimizador 1 ha terminado, se le envía el nuevo proceso de optimización.
    - Si no ha terminado, el proceso de optimización se envía al optimizador 2.

- Si la anterior optimización se envió al optimizador 2, se debe ver si éste ha terminado.
    - Si el optimizador 2 ha terminado, se le envía el nuevo proceso de optimización.
    - Si no ha terminado, el proceso de optimización se envía al optimizador 1.

A pesar de la planificación la disposición de dos únicos optimizadores y las restricciones antes mencionadas hacen el tiempo de respuesta de un problema de optimización sea relativamente amplio. Por ello, se ha diseñado una funcionalidad que consiste en consultar el estado de un proceso de optimización. Bien cada cierto intervalo de tiempo, o bien cuando el usuario lo crea oportuno, se puede consultar el estado en el que se encuentran los procesos que ha enviado a optimizar un cliente.

Estados de un Proceso de Optimización

Desde que el usuario crea un proceso de optimización, seleccionando una base de datos con la topología del problema hasta que llega a la aplicación cliente la solución a la petición de optimización, dicho proceso de optimización pasa por una serie de estados, que se distribuyen en el cliente, servidor y optimizador. Los estados por los que pasa un proceso son los siguientes:

- Estados en la aplicación cliente:
    - Estado 0 => *CREADO.* Tiene lugar cuando el usuario selecciona una base de datos. En este momento se crea un proceso de optimización.
    - Estado 9 => *FINALIZADO. PROCESO EN EL CLIENTE.* El cliente ha recibido la solución de la optimización de manos del planificador. Se puede decir que en este momento, el tiempo de vida del proceso de optimización ha finalizado.

- Estados en el servidor:
    - Estado 1 => *ENVIADO AL PLANIFICADOR*. El cliente ha enviado el proceso de optimización para que realice la optimización del problema. Este estado se produce cuando el proceso ha quedado registrado en la base de datos del planificador.
    - Estado 2 => *ENVIADO PARA OPTIMIZAR*. Una vez realizada la planificación se conoce a qué optimizador se va a enviar el problema. Este estado tiene lugar cuando se hace la llamada al optimizador correspondiente para que resuelva el problema.
    - Estado 8 => *SOLUCION CREADA. PLANIFICADOR DEVOLVIENDO SOLUCIÓN*. Cuando el planificador recibe la solución de manos del optimizador, le reenvía dicha solución al cliente. Este estado representa esta situación.

- Estados en el optimizador:
    - Estado 3 => *EN OPTIMIZADOR. ESPERANDO PARA EJECUTAR*. El problema de optimización ha llegado al optimizador. Aún no ha comenzado su ejecución.
    - Estado 4 => *RELLENANDO TABLAS PARA EJECUTAR.* El optimizador utiliza una serie de tablas para llevar a cabo la optimización del problema. En este estado, la información sobre el problema que lleva el DataSet se pasa a dichas tablas.
    - Estado 5 => *EJECUTANDO EL ALGORITMO GENÉTICO*. En este estado es donde se realiza la optimización propiamente

dicha. Finaliza cuando se devuelve la solución a la optimización. Posiblemente sea el estado con mayor carga de trabajo.

- Estado 6 => *EJECUCIÓN TERMINADA. CREANDO SOLUCIÓN.* Una vez que el algoritmo ha creado una solución, se deben devolver unas tablas con dicha solución. En este estado se rellenan dichas tablas.
- Estado 7 => *SOLUCIÓN CREADA. OPTIMIZADOR DEVOLVIENDO SOLUCIÓN.* La solución ya está en las tablas. Ahora el optimizador devuelve el DataSet con dicha solución al planificador.

La actualización de los estados de un proceso de optimización debe quedar reflejada en la base de datos del planificador. Es claro que puede haber varios procesos optimizando, y en cada momento se debe conocer cuál es el proceso que ha cambiado de estado. La manera de identificar unívocamente a cada proceso es mediante el identificador del cliente (número entero) y el nombre del proceso (string).

Funcionalidad

Funciones Disponibles

Se puede decir que el planificador es una estructura cerrada, poco afectada por eventos. Posee dos funcionalidades básicas que son:

1. **Dar de alta a clientes.**

   El planificador conserva un registro de todos los clientes que han hecho uso de la aplicación. Si un cliente es nuevo debe registrarse. Para ello, el servidor crea un identificador y una cadena de caracteres con el nombre del cliente, que se devuelve al usuario.

2. **Realizar una planificación**

   Cuando el cliente envía un proceso para optimizar, el servidor activa la planificación, para enviar el problema a un optimizador. Posteriormente devuelve el resultado generado por dicho optimizador.

3. **Consultar el estado de un proceso de optimización**

   Cuando el proceso de optimización se está ejecutando, el cliente puede pedirle al servidor que le diga cuál es el estado actual de dicho proceso. El servidor devolverá dicho estado.

4. **Actualizar el estado de un proceso de optimización**

   El estado de un proceso de optimización puede cambiar en el cliente, en el servidor o en el optimizador. El servidor debe proporcionar una manera de modificar dicho estado.

Para llevar a cabo la planificación del optimizador al que se va a enviar el proceso de optimización, el servidor realiza algunas funciones internas.

Primero realiza la planificación propiamente dicha, donde se elige el optimizador al que se va a enviar la optimización. Posteriormente da de alta al proceso de optimización en su base de datos, guardando los atributos necesarios. A continuación envía la información necesaria al optimizador seleccionado y por último devuelve el resultado al cliente.

Mejoras

En relación a la planificación se podría haber optado por tener un número mayor de optimizadores. Siguiendo con la política de turno rotatorio, al aumentar el número de optimizadores, se mejora el tiempo de respuesta. Sin embargo, el objetivo de la aplicación no es minimizar el tiempo de respuesta de una optimización sino establecer las comunicaciones entre distintos módulos mediante servicios web. Por esta razón se ha preferido utilizar tan solo dos optimizadores.

También podría optarse por mejorar la política de planificación si se permitiera interrumpir un proceso de optimización para atender otro, o aún mejor si dos optimizadores pueden trabajar en una misma optimización. También podría considerarse la mejora de estimar el menor tiempo restante de optimización en el caso de que ambos optimizadores estén ocupados. Nuevamente esto se sale de los objetivos de la aplicación.

Una mejora que sí se puede tener en cuenta en próximas versiones es la posibilidad en enviar simultáneamente varios procesos de optimización, en lugar de uno cada vez. En este caso el servidor recibiría un lote de problemas a optimizar, que debería planificar y enviar al optimizador correspondiente.

Se pueden añadir más estados para un problema de optimización. Esto sobre todo tiene sentido dentro de la ejecución del algoritmo genético, ya que es la parte con mayor carga de trabajo. Así se consigue una mayor precisión al definir la situación del estado actual de cada proceso de optimización.

Se podrían incluir consultas sobre el estado de la red o configurarla para mejorar el envío de información.

Arquitectura del Servidor

Esquema General y Partes del Servidor

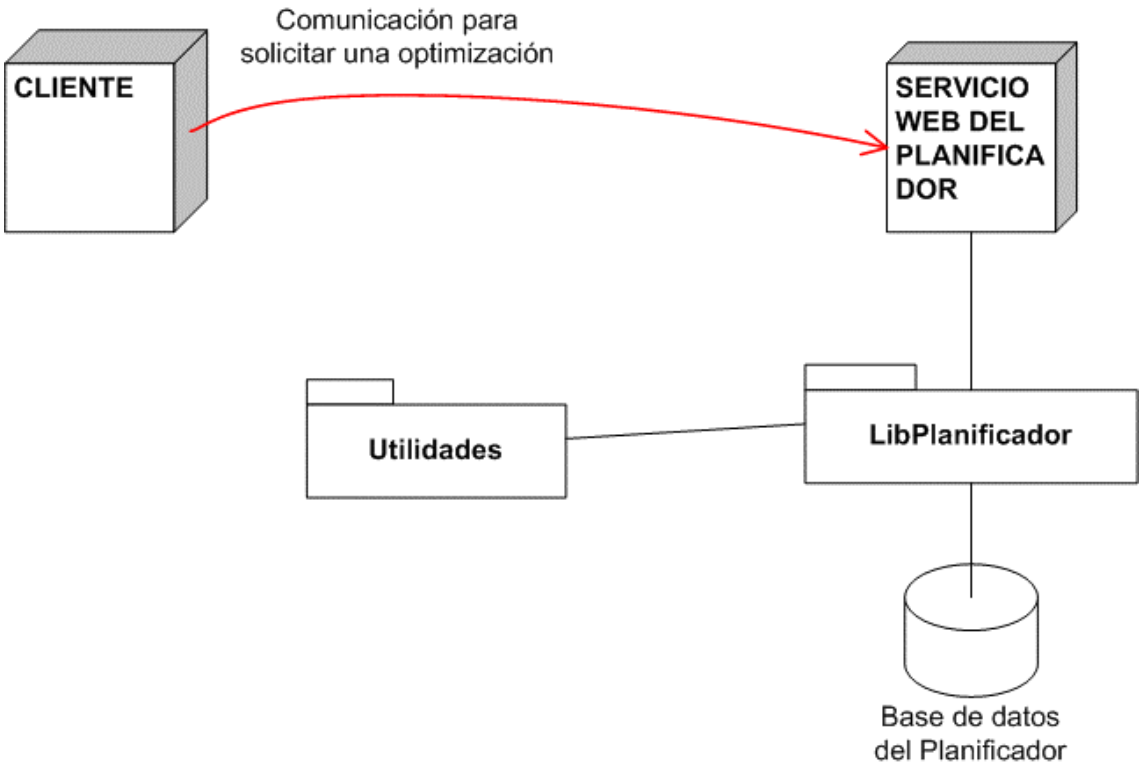
La aplicación está compuesta esencialmente por un servicio web que se comunica con el cliente y que proporciona los datos necesarios para que el servidor realice la planificación, envíe el problema a un optimizador y devuelva los datos del resultado al cliente. También proporciona servicio para consultar el estado de los procesos de optimización, mediante el identificador del cliente y del proceso; y para registrar a los nuevos clientes, devolviéndoles el nombre de su identificación.

Este servicio web se apoya en una librería de clases que es la encargada de realizar todas las funciones internas del servidor. A su vez, esta librería está enlazada a una base de datos del planificador, donde se guardan todas las variables necesarias para la planificación, el registro de clientes y los procesos de optimización que recibe.

Para llevar la gestión sobre la base de datos se hace uso de una clase que se encuentra en la librería de clases llamada *Utilidades*. Esta clase se llama *BaseDatos* y proporciona métodos que facilitan consultas sobre una base de datos; cargan bases de datos en una estructura DataSet o guarda el contenido de un DataSet en una base de datos. Se importa dicha librería.

Por tanto en esencia, el servidor se puede dividir en dos partes: Servicio web y librería. La primera simplemente se utiliza como interfaz con el cliente, mientras que la segunda implementa toda la funcionalidad interna del servidor.

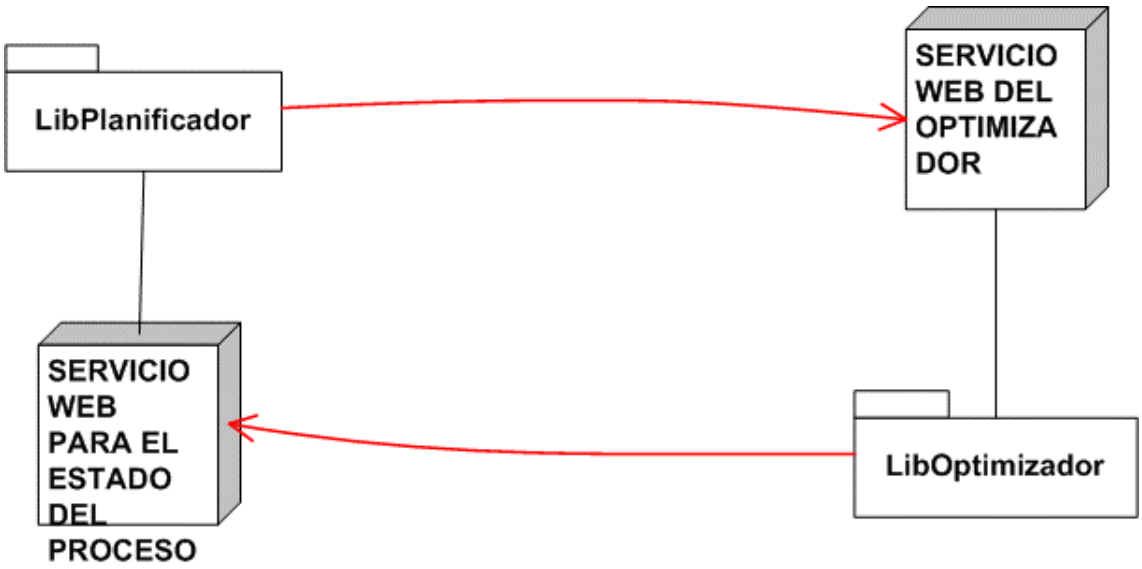
A continuación se muestra un esquema de la arquitectura básica del servidor:



El planificador a su vez debe comunicarse con los optimizadores. Esto lo realiza también a través de servicios web. Sin embargo, la comunicación entre el servidor y los optimizadores es bidireccional.

- Por un lado, se encuentra la comunicación *servidor -> optimizador*. Tiene lugar cuando se pide al optimizador que realice la optimización del problema. El servidor envía al optimizador seleccionado los datos necesarios, y éste realiza la optimización. El encargado de realizar estas peticiones es la librería del planificador y tienen efecto sobre la librería del optimizador.
- Por otro lado, se encuentra la comunicación *optimizador -> servidor*. Se produce para actualizar el estado del proceso de optimización dentro del optimizador; y para indicar si el optimizador está o no activo. Estas peticiones las realiza la librería del optimizador y tienen efecto sobre la librería del planificador.

Se muestra un pequeño esquema de la comunicación entre el servidor y un optimizador:



Clases Implementadas

SWPlanificador

Como ya se ha dicho, se trata del servicio web que comunica el cliente con el servidor. Posee una variable privada que hace referencia a la clase principal de la librería del planificador:

- *private ClasePlanificador planificador*

Posee a su vez cuatro *WebMethods* que llaman a los métodos de la librería del planificador.

- *[WebMethod] public string addCliente()*:

  Realiza una llamada al método con el mismo nombre del planificador, y devuelve una cadena con el nombre del cliente.

- *[WebMethod] public DataSet ejecutar(string idC, string nombre, DataSet DS, string BD):*

  Realiza una llamada al método con el mismo nombre del planificador, y devuelve el resultado de la optimización, que se codifica en una estructura DataSet.

  - String idC: Es el identificador del cliente. Representa el nombre que anteriormente le ha proporcionado el servidor al cliente. Se utiliza para las consultas sobre la base de datos del planificador.
  - String nombre: Es el nombre del proceso de optimización. Éste nombre se lo proporciona la aplicación cliente cuando se quiere optimizar un problema y se le asigna una base de datos con la topología adecuada.
  - DataSet DS: Es una estructura DataSet que guarda las tablas con el problema a resolver. Adicionalmente lleva una tabla con la configuración que el cliente ha decidido sobre las características de la optimización.
  - String BD: Es la dirección donde se encuentra la base de datos que representa el problema de optimización de la red de transportes.

- *[WebMethod] public string getEstado(string idC, string nombre):*

  Realiza al servidor una petición sobre el estado actual del proceso de optimización que se identifica unívocamente mediante el identificador del cliente y el nombre del proceso. Devuelve una cadena con el estado actual de la optimización.
  - String idC: Cadena que representa al identificador del cliente.
  - String nombre: Cadena que representa al nombre del proceso por el que se pregunta el estado.

- *[WebMethod] public string setGetEstado(string idC, string nombre, int estado):*

  Dentro de la aplicación cliente también se realizan cambios de estado de un proceso de optimización. Este método hace que el servidor actualice el estado del proceso identificado a través de los dos primeros parámetros con el estado que representa el tercer parámetro. Posteriormente se devuelve la cadena correspondiente a dicho estado, para hacérsela saber al cliente. En el cliente se actualizan el primer y último estado de un proceso de optimización, ya que es ahí donde comienza y termina su tiempo de vida.
  - String idC: Cadena que representa al identificador del cliente.
  - String nombre: Cadena que representa al nombre del proceso sobre el que se va a actualizar el estado.
  - Int estado: Codificación del estado que se quiere actualizar.

LibPlanificador.ClasePlanificador

Esta clase se compone de la parte funcional del servidor. Accede a la base de datos para registrar nuevos clientes, para añadir nuevos procesos de optimización o para actualizar los atributos necesarios para gestionar el algoritmo de planificación, así como para realizar consultas sobre algunas variables de registros. Planifica cada proceso que le llega, asignándole un optimizador y manda a ejecutar dicho proceso. También realiza consultas sobre el estado del proceso, actualizándolo cuando sea necesario. Tiene una referencia al servicio web del optimizador y utiliza la clase que gestiona una base de datos de la librería de utilidades.

En cuanto a los atributos de la clase, posee dos referencias web a sendos optimizadores, así como una instancia de la clase *BaseDatos* de la librería *Utilidades*, que facilita enormemente el acceso y actualización de la base de datos del planificador. Por último posee una cadena que representa la dirección de esta base de datos.

- *private SWOptimizador op1*
- *private SWOptimizador op2*
- *private BaseDatos bd*
- *private static string bdPlanificador = @"C:\Inetpub\wwwroot\SIG2\ SWPlanificador\BDPlanificador.mdb"*

Para llevar a cabo todas sus funcionalidades posee una serie de métodos que se pasan a comentar a continuación:

- *private int planificar() :*

  Método que realiza la planificación del proceso que se va a mandar a optimizar. Devuelve un entero que representa el optimizador que va a llevar a cabo la optimización. Realiza la planificación descrita anteriormente, consultando en la tabla Planificador de la base de datos cuál fue el último optimizador al que se llamó para realizar una optimización y si se encuentra activo actualmente. Posteriormente actualiza el valor que representa el último optimizador al que se llamó, con el valor del que ha resultado de la planificación.

- *public DataSet ejecutar(string idCliente, string idProceso, DataSet ds, string BD):*

  Método que realiza la ejecución del algoritmo de optimización. Primero realiza la planificación, obteniendo el optimizador al que se debe mandar

el problema. A continuación almacena en la base de datos un nuevo registro para el proceso de optimización solicitado. Esta operación se realiza llamando a otro método de la clase. Seguidamente, dependiendo del resultado de la planificación, se envía el problema al optimizador. En este momento, el proceso se encuentra ejecutándose en el optimizador. Cuando llega la solución, se actualiza el estado del proceso, que ya se devolverá al cliente. Esto es precisamente lo que devuelve el método; una estructura DataSet con la solución al problema de optimización.

- string idCliente: Es una cadena con el identificador del cliente. Sirve para obtener de la tabla Clientes el identificador del cliente (entero). Este dato se envía al optimizador para identificar unívocamente el proceso de optimización que está tratando.
- String idProceso: Nombre del proceso. Se utiliza para registrar un nuevo proceso de optimización en la base de datos, y para enviarlo al optimizador. De esta manera se identifica unívocamente el proceso de optimización a la hora de realizar consultas.
- DataSet ds: Es una estructura DataSet con la información del problema a optimizar. Se envía al optimizador para que resuelva dicho problema.
- String BD: Es la base de datos asociada al problema de optimización. Se guarda esta información al dar de alta un nuevo problema de optimización en la tabla Procesos.

- *public string addCliente():*

  Añade un nuevo cliente en la base de datos del planificador. Se añade en la tabla Clientes. Primero se consulta sobre el número de registros que tiene la tabla (el primer registro se identifica como 0). Este número será el que identifique al cliente. El nombre del cliente se crea mediante la cadena '*"Cliente_"* + *número*'. El método devuelve el nombre del cliente, para hacérselo saber al usuario.

- *public void addProceso(string idCliente, string nombreProceso, string BD, int opt):*

  Método que añade un nuevo proceso de optimización a la base de datos del planificador. Este nuevo registro se añade en la tabla Procesos. Esto sucede cuando el cliente manda que se optimice el problema. Primero, utilizando el nombre del cliente, se obtiene el identificador del cliente de la tabla Clientes. Una vez que se tiene este identificador, se realiza una segunda consulta sobre la base de datos. Esta vez sobre la tabla Procesos, se obtiene el número de registros que tiene el cliente con el identificador obtenido. Esto equivale a obtener el número de procesos de optimización que ha enviado dicho cliente. Este valor va a coincidir con el identificador (entero) del nuevo proceso que se va a registrar en la base de datos. Una vez que se han conseguido todos los datos se crea

un nuevo registro y se añade a la tabla Procesos. Por último se actualiza la base de datos del planificador.

- String idCliente: Es la cadena que identifica al cliente que realiza la petición de optimización. Esta variable se utiliza para consultar en la tabla Clientes.
- String nombreProceso: Nombre del proceso que se va a dar de alta en la base de datos. Este dato lo proporciona el cliente.
- String BD: Representa la dirección de la base de datos que representa la red de transportes a optimizar. También se añade como dato en el registro del proceso.
- Int opt: Es un valor que identifica al optimizador que va a realizar la optimización del problema.

- *public void setEstado(string idC, string idP, int estado):*

  Método que actualiza el estado de un proceso. Esta actualización se realiza sobre la tabla Procesos de la base de datos. Primero se realiza una consulta sobre la tabla Cliente para conseguir el identificador (entero) del cliente, conociendo su nombre. Sobre la tabla de procesos, se selecciona el registro cuyo identificador de cliente y nombre de proceso coincidan con los dados. Se actualiza el campo que representa el estado del proceso con la variable que se pasa como parámetro. Por último se actualiza la base de datos con los cambios.

  - String idC: Nombre del cliente. Se va a utilizar para adquirir el identificador del cliente en la tabla Clientes.
  - String idP: Nombre del proceso sobre el que se va a actualizar el estado.
  - Int estado: Número entero que representa el estado que se va actualizar

- *public void setActivo(int idOpt, int activo):*

  Realiza una actualización sobre la tabla Planificador, indicando si el optimizador que se pasa como parámetro está o no activo. Se obtiene el primer registro de la tabla Planificador y se cambia el campo 'idOpt' ('1' para el primero optimizador y '2' para el segundo) con el valor 'activo' ('0' si no está activo y '1' si sí lo está). Posteriormente se actualiza la base de datos con los cambios realizados.

  - int idOpt: Es un entero que identifica al optimizador.
  - Int activo: Variable que indica si el optimizador en cuestión está o no activo.

- *public string getEstado(string idC, string idP):*

  Método que devuelve el estado del proceso que se identifica con los parámetros de entrada. Primero se realiza una consulta sobre la tabla Planificador donde se obtiene el identificador del cliente a partir de su nombre. A continuación se realiza una segunda consulta, esta vez sobre la tabla Procesos, donde se recoge el estado del proceso cuyo identificador de cliente y nombre de proceso coinciden con los dados. En este momento se tiene una codificación numérica del estado actual del proceso de optimización requerido. Se realiza una llamada a un método interno que transforma este código a una cadena de caracteres que indica el estado actual del proceso. Esta cadena es el valor que se devuelve.

  - string idC: Nombre del cliente.
  - String idP: Nombre del proceso sobre el que se quiere consultar el estado actual.

- *private string getCadena(int estado):*

  Método que genera la cadena del estado del proceso, según la codificación que se pasa en el parámetro de entrada. Devuelve dicha cadena.

  - Int estado: Entero que representa la codificación del estado de un proceso de optimización. Esta codificación se obtiene de la base de datos.

SWOptimizador

Este servicio enlaza el servidor con el optimizador. Se encarga de hacerle llegar la información al algoritmo de optimización y de que éste comience su ejecución. Su única variable hace referencia a una instancia de la clase *ClaseAlgoritmoGenetico*.

- *private ClaseAlgoritmoGenetico algoritmo.*

Igual que sucede con el servidor, este servicio web sirve únicamente como interfaz para el optimizador. Ya que la funcionalidad del algoritmo de optimización se encuentra en la librería del optimizador. Sólo posee un método para enviarle los parámetros al optimizador, y devuelve el resultado al servidor.

- *[WebMethod] public DataSet ejecutarAlgoritmo(string idC, string idP, DataSet DS, int id):*

  Crea la instancia del algoritmo genético llamando a la constructora con los parámetros de la interfaz y manda ejecutar dicho algoritmo. Devuelve un DataSet con el resultado de la optimización.

  - String idC: Nombre del cliente que ha pedido la solicitud de optimización.
  - String idP: Nombre del proceso que se va a optimizar.
  - DataSet DS: Estructura de datos que contiene la información del problema a optimizar, recogido de la base de datos que representa la red de transportes. Además contiene una tabla con los datos de configuración del algoritmo.
  - Int id: Es el identificador del optimizador. Dependiendo de a qué optimizador haya correspondido realizar la operación, puede tomar los valores 1 o 2.

LibOptimizador.ClaseAlgoritmoGenetico

Sobre esta clase, se van a comentar los métodos que guardan relación con el servidor, que van a ser los encargados de mantener el estado del proceso que se está ejecutando, y de actualizar la variable que indica si el optimizador está o no activo. Posee una referencia al servicio web que modifica los datos anteriormente citados.

La clase posee una serie de variables estáticas que simplemente identifican los estados que tiene lugar dentro del optimizador. Posee una variable privada que es un DataSet, que almacena el problema a optimizar. Además posee variables para el nombre del cliente y del proceso sobre el que se va a realizar la optimización. Estas variables son fundamentales para conocer el proceso concreto a la hora de actualizar el estado. Con un propósito parecido, posee una variable que identifica al optimizador (1 o 2). Esta variable se utiliza para actualizar la variable que indica si el optimizador esta o no activado.

- *private static int SINEMPEZAR = 2*
- *private static int RELLENO = 3*
- *private static int EJECUCION = 4*
- *private static int DEVOLVIENDO = 5*
- *private static int TERMINADO = 6*
- *private DataSet ds*

- *private string idC*
- *private string idP*
- *private int ID*

Los métodos que guardan relación con el servidor son:

- *public ClaseAlgoritmoGenetico(string idCliente, string idProceso, DataSet DS,int id):*

  Constructora con parámetros de la clase. Asigna los valores que se pasan como parámetros a las variables privadas correspondientes.

  - String idCliente: Nombre del cliente que pide la solicitud de optimización
  - String idProceso: Nombre del proceso que se va a optimizar. Junto con el parámetro anterior, sirve para identificar unívocamente al proceso de optimización de forma que la actualización del estado se realiza de forma sencilla sobre el servidor pasando estos dos parámetros.
  - DataSet DS: Estructura que contiene el problema a optimizar, junto con la configuración del algoritmo genético.
  - Int id: Entero que identifica al optimizador. Se utiliza para modificar en el servidor la variable que indica si el optimizador está o no activo.

- *private void sendEstado(int estado):*

  Se trata de un método privado que envía el nuevo estado al servidor en el transcurso de la optimización. Crea una instancia del servicio web que ofrece el servicio de actualizar el estado y envía al servidor, a través de ese servicio web el nuevo estado, junto con la identificación unívoca del proceso que se está optimizando.

  - Int estado: Entero que codifica el estado actual del proceso de optimización

- *private void sendActivo(int activo):*

  Es un método privado que envía al servidor la actualización de la variable que indica si el optimizador está o no activo. Al comienzo de la optimización se enviará como ‘activo’, y al finalizar se enviará como ‘no activo’. El método crea una instancia del servicio web que ofrece el servicio de actualizar esta variable y envía al servidor, a través de dicho servicio web esta actualización, junto con el identificador del optimizador.

  - Int activo: Variable que identifica el flag de actividad del optimizador.

SWEstadoProceso

Este servicio web comunica el optimizador con el servidor. Sirve de enlace entre ambos para actualizar el estado de un proceso que se está optimizando y para actualizar el *flag* de actividad del optimizador. Posee una referencia a la librería del planificador. Su única variable es una instancia del planificador.

- *private ClasePlanificador planificador.*

Como se ha dicho anteriormente, posee dos métodos que enlazan con el servidor:

- [WebMethod] public void setEstado(string idC, string idP, int estado)

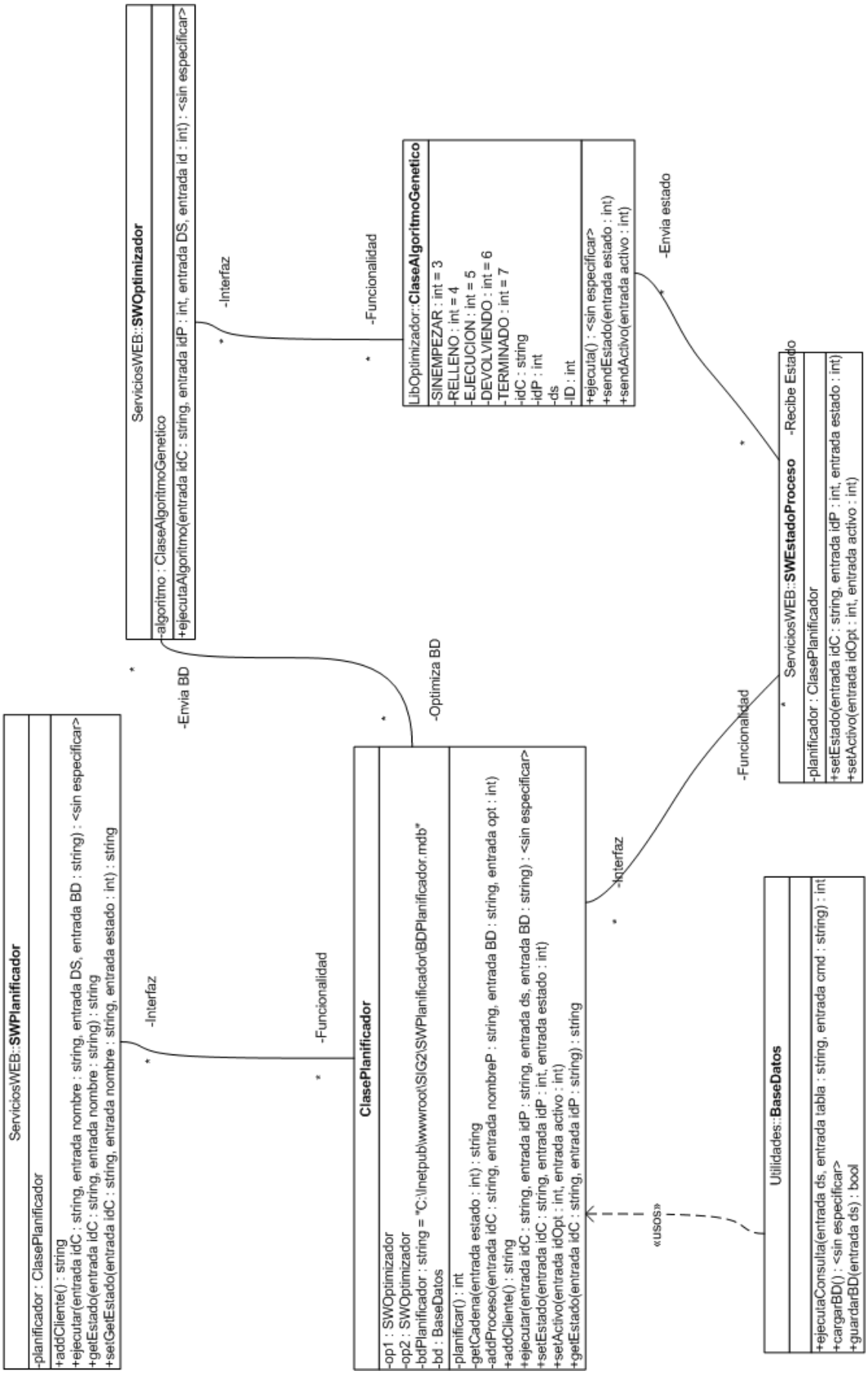
  Llama al método con el mismo nombre del planificador, pasándole los parámetros de la interfaz.
  - String idC: Nombre del cliente poseedor del proceso que se está optimizando.
  - String idP: Nombre del proceso que se está optimizando.
  - Int estado: Nuevo estado del proceso, codificado mediante un número entero.

- *[WebMethod] public void setActivo(int idOpt, int activo):*

  Llama al método del mismo nombre del planificador, pasándole los parámetros de la interfaz
  - Int idOpt: Identificador del optimizador.
  - Int activo: Flag de actividad del optimizador

Diagrama de Clases UML del Servidor

En la página siguiente se especifica en detalle un diagrama de clases de la aplicación servidor. En él se pueden ver las diferentes clases que intervienen en este módulo, sus atributos y sus métodos; así como las relaciones entre ellas:


ServiciosWEB::SWPlanificador
-planificador : ClasePlanificador
+addCliente() : string
+ejecutar(entrada idC : string, entrada nombre : string, entrada DS, entrada BD : string) : <sin especificar>
+getEstado(entrada idC : string, entrada nombre : string) : string
+setGetEstado(entrada idC : string, entrada nombre : string, entrada estado : int) : string
-Interfaz
-Funcionalidad
ClasePlanificador
-op1 : SWOptimizador
-op2 : SWOptimizador
-bdPlanificador : string = "C:\Inetpub\wwwroot\SIG2\SWPlanificador\BDPlanificador.mdb"
-bd : BaseDatos
-planificar() : int
-getCadena(entrada estado : int) : string
-addProceso(entrada idC : string, entrada nombreP : string, entrada BD : string, entrada opt : int)
+addCliente() : string
+ejecutar(entrada idC : string, entrada idP : string, entrada ds, entrada BD : string) : <sin especificar>
+setEstado(entrada idC : string, entrada idP : int, entrada estado : int)
+setActivo(entrada idOpt : int, entrada activo : int)
+getEstado(entrada idC : string, entrada idP : string) : string
-Interfaz
«uses»
Utilidades::BaseDatos
+ejecutaConsulta(entrada ds, entrada tabla : string, entrada cmd : string) : int
+cargarBD() : <sin especificar>
+guardarBD(entrada ds) : bool
-Envia BD
-Optimiza BD
ServiciosWEB::SWOptimizador
-algoritmo : ClaseAlgoritmoGenetico
+ejecutaAlgoritmo(entrada idC : string, entrada idP : int, entrada DS, entrada Id : int) : <sin especificar>
-Interfaz
-Funcionalidad
LibOptimizador::ClaseAlgoritmoGenetico
-SINEMPEZAR : int = 3
-RELLENO : int = 4
-EJECUCION : int = 5
-DEVOLVIENDO : int = 6
-TERMINADO : int = 7
-idC : string
-idP : int
-ds
-ID : int
+ejecuta() : <sin especificar>
+sendEstado(entrada estado : int)
+sendActivo(entrada activo : int)
-Envia estado
-Funcionalidad
ServiciosWEB::SWEstadoProceso
-Recibe Estado
-planificador : ClasePlanificador
+setEstado(entrada idC : string, entrada idP : int, entrada estado : int)
+setActivo(entrada idOpt : int, entrada activo : int)

Optimizador

Descripción General

El optimizador es un módulo que implementa un algoritmo genético que resuelve el problema de optimización de la red de transportes. Además de ir resolviendo los problemas que le llegan del servidor, irá actualizando el estado en el que se encuentra dicho problema. Esto es, en cada fase del algoritmo (recogida de datos, creación de la población, comprobación de las soluciones obtenidas, generación de la solución...) el optimizador hará saber al servidor el estado concreto en el que se encuentra el problema que envió para resolver.

La información que se transmite entre el servidor y el optimizador se codifica en un DataSet. El optimizador lo recibe, de él lee los datos que los guarda en sus tablas y, tras encontrar la solución, añade al DataSet de entrada las nuevas tablas que conforman la solución.

Además, existe una segunda conexión, esta vez en sentido optimizador -> servidor, que sirve para actualizar el estado de las optimizaciones, así como el *flag* de activación del algoritmo genético.

Estos enlaces entre el servidor y el optimizador se realiza mediante servicios web.

En cuanto al algoritmo en sí, un rasgo importante a tener en cuenta es la utilización del elitismo para conseguir la mejor solución. El elitismo se ha utilizado para seleccionar cuales son los mejores individuos de la población en cada iteración y así ni cruzarlos, ni mutarlos ni seleccionarlos, con el correspondiente riesgo de que no sean elegidos, normalmente el porcentaje de individuos que se introducen en la élite suele ser de un 10% del total de la población.

Para la aplicación va a existir dos optimizadores; cada uno en una máquina distinta. El encargado de seleccionar qué optimizador resuelve qué problema de optimización es el planificador.

Funcionalidad

Funciones Disponibles

De cara a la comunicación con el servidor, el optimizador puede realizar las siguientes funciones:

1. **Ejecutar un problema de optimización.**

   El planificador envía al optimizador el problema a resolver, junto con otros datos relevantes a la hora de actualizar el estado del problema y la activación del algoritmo. Internamente, el optimizador realizará las operaciones oportunas para devolver la solución.

2. **Actualizar el estado del problema de optimización**

   El algoritmo genético se puede dividir en una serie de partes. Cada una de ellas constituye un estado en el que puede encontrarse un problema de optimización. El optimizador debe informar al servidor de cualquier cambio de estado del proceso que está optimizando.

3. **Actualizar el *flag* de actividad del algoritmo**

   Debido a la planificación, el servidor necesita conocer si los optimizadores están o no realizando una optimización. Por ello, el algoritmo debe informar al servidor sobre esta situación. Cuando comience una optimización, el optimizador indicará que está *"activo"*, y cuando la finalice y devuelva el resultado, indicará que está *"no activo"*.

Mejoras

La generación de individuos puede realizarse, haciendo subindividuos, es decir, dependiendo de cada producto, si un producto es válido, se mantiene y se intentan generar los demás productos, por el contrario tal y como está implementado, se desecha el individuo y se genera uno nuevo, perdiendo la información válida que hay.

Arquitectura del Optimizador

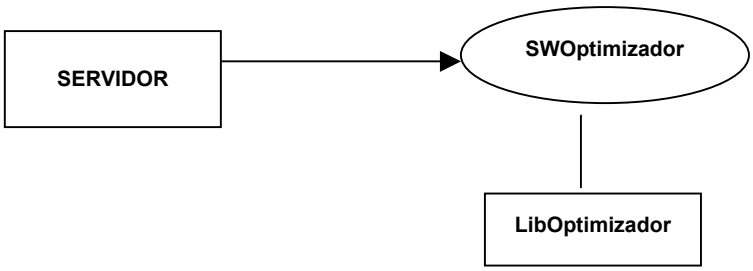
Esquema General y Partes del Optimizador

La aplicación está compuesta por un servicio web que se comunica con el servidor y que proporciona los datos de la optimización y de la identificación del cliente y proceso de optimización.

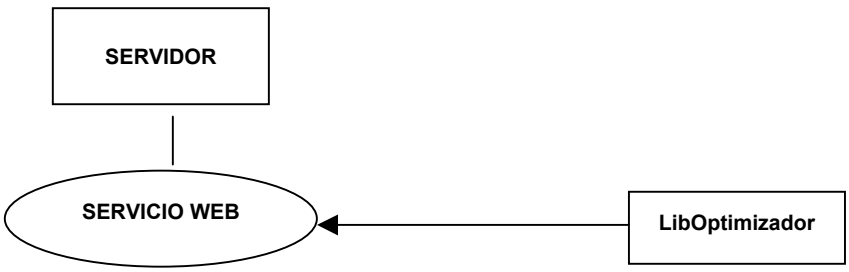
Este servicio se apoya en una librería que codifica la parte funcional del optimizador, es decir, implementa el algoritmo genético y los métodos necesarios para la actualización del estado y del flag de activación. Está compuesta por una serie de clases que implementan dicho algoritmo. Estas clases pasarán a comentarse más adelante.

Por tanto, se puede decir que el optimizador está dividido en dos partes. El servicio web que realiza la comunicación con el servidor, y transmite los datos necesarios; y la librería, que realiza todas las funcionalidades internas del algoritmo de optimización.

A continuación se muestra un esquema de la arquitectura básica del optimizador:



Además, el optimizador tiene un enlace a un servicio web del servidor para poder actualizar el estado de las optimizaciones y del flag de activación. Éste es el esquema:



Clases Implementadas

El optimizador ha sido desarrollando utilizando la tecnología C#, la misma que para el resto del proyecto, además de XML para enviar y recibir la base de datos en la comunicación con el planificador. Se trata de una librería de clases de C# llamada: *LibOptimizador*.

Está estructurado en clases: la clase Individuo, población, celda, celda_demanda y algoritmo genético, propiamente dicho, en el cual se realizan las operaciones de creación de la población, mutación, cruce…

A continuación se procede a describir estas clases:

Clase Celda

La clase Celda se encarga de almacenar la información que recibe de la base de datos, de las tablas Transporte1, Transporte2 y Suministro.

Los atributos de la clase celda son:

- *Origen*: indica el origen del producto.
- *Destino*: indica el destino del producto.
- *Producto*: indica el identificador del producto.
- *Cantidad*: cantidad máxima de producto.
- *Coste*: Coste de transportar cada unidad de producto.

Esta clase rellena los objetos completos cuando se trata de las tablas Tranporte1 y Tansporte2, porque tienen todos los campos, en cambio, la tabla Suministro carece de campo origen, así para reutilizar esta clase, se ha decidido rellenar el campo origen como 0 para dicha tabla.

Clase Celda_demanda

La clase Celda_demanda se encarga de almacenar la información que recibe de la base de datos, en concreto de la tabla Demanda.

Los atributos de la clase celda son:

- *Nombre*: nombre del demandante.
- *Producto*: indica el identificador del producto.
- *Cantidad*: cantidad demandada del producto de producto.

Clase individuo

Esta es la clase encargada de generar los individuos y de mantener toda la información referente a cada uno de los individuos creados

Los atributos de la clase Individuo son:

- *Aptitud*: función que calcula el coste del individuo
- *Icrom1*: elementos del primer nivel de transporte.
- *Icrom2*: elementos del segundo nivel de transporte.
- *Isum*: número de suministradores disponibles.
- *esvalido*: booleano que indicará si es un individuo válido o no.
- *puntuación*: parámetro del algoritmo genético.
- *punt_acu*: parámetro del algoritmo genético.
- *t1*: array que contiene valores entre 0 y 1, correspondientes a la cantidad de producto del primer nivel de transporte, que cogemos para dicha arista, las cuales se identificación por su posición en el array, y en el grafo de la base de datos de izquierda a derecha.
- *t2*: array que contiene valores entre 0 y 1, correspondientes a la cantidad de producto del segundo nivel de transporte, que cogemos para dicha arista, las cuales se identificación por su posición en el array, y en el grafo de la base de datos de izquierda a derecha.
- *s*: array que contiene valores aleatorios entre 0 y 1, correspondientes a la cantidad de producto que cogemos de cada suministrador.

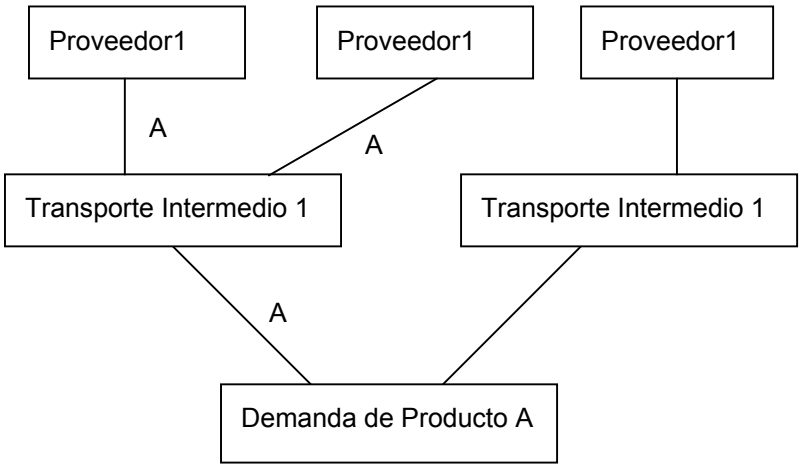
El funcionamiento de esta clase es el siguiente.

La clase individuo funciona de la siguiente manera: Se generan aleatoriamente los arrays t1, t2 y s. A continuación se calcula la aptitud del

individuo. Este parámetro consiste en una función que calcula el coste del individuo. Además calcula si es válido o no dicho individuo aleatorio generado, basado en si se satisfacen o no todas las demandas, permitiendo un margen de 50 elementos. Es decir, se podrían enviar más demandas de las necesarias, principalmente para no ralentizar en exceso el algoritmo genético, dado que si exigimos una optimización muy estricta, los tiempos de ejecución serían muy elevados.

Probablemente la parte más compleja del optimizador, se encuentra en esta clase, que corresponde al método de función, el cual calcula el coste de un individuo, es un algoritmo cúbico $O(n^3)$.

A continuación, mostraremos un ejemplo gráfico de cómo funciona dicho algoritmo.



Para cada línea de producto A dirigido a cada demandante se buscan hacia atrás todas las aristas que confluyen el transporte intermedio para dicho producto, en este caso las líneas azules

Clase Población

La clase población, es la encargada de gestionar el comportamiento de los individuos, esta clase tiene los métodos característicos de todo algoritmo genético. Los métodos de selección, reproducción, evaluación, mutación y cruce, además tiene varios métodos de comprobación de la validez de los individuos.

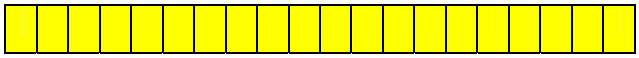
El método selección

Este método es el encargado de elegir los miembros de la población que van a sobrevivir, los que no son seleccionados son eliminados y sustituidos por unos nuevos.

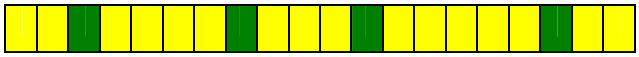
El método mutación

Este método es el encargado de modificar los individuos válidos generados hasta el momento, la táctica para modificar dichos elementos es la siguiente, la probabilidad de mutación, es un atributo del algoritmo genético, se genera un número aleatorio, si es menos que la probabilidad de cruce, se modifica dicho archivo, sino lo es, entonces se continúa, a continuación voy a mostrar con un ejemplo gráfico como funciona la mutación. Cada una de las ranuras de este *"cromosoma"*, se corresponde con un valor del "gen", es decir, serían valores aleatorios entre 0 y 1, para cada uno de los valores, se calcula una probabilidad, si es menor que la probabilidad de mutación del algoritmo, entonces se genera un nuevo valor.

Antes:



Después:



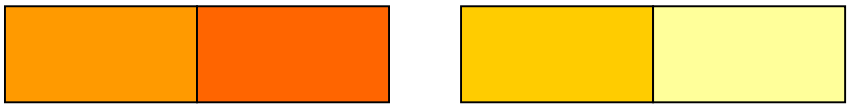
Las casillas en verde, son los nuevos valores para esas posiciones del array.

El método reproducción

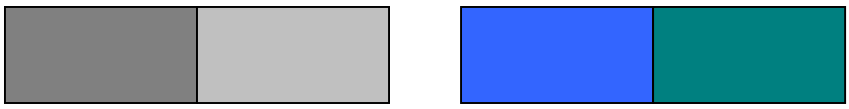
Este método es el encargado de cruzar los individuos entre sí, en este algoritmo genético, se ha modificado un poco el típico cruce de otros algoritmos genéticos, debido a que cada individuo tiene, por decirlo así tres vectores de genes, creados de forma aleatoria, en el momento de la generación de los individuos. Así cuando cruzamos dos individuos, se obtienen los siguientes resultados que voy a mostrar de forma gráfica:

La línea de separación entre las dos partes de cada vector, se corresponde con el punto de cruce obtenido de forma aleatoria, así que aunque en este caso esté situado en el centro, no tiene porque estar en dicha posición.

Padre 1: Vectores de transporte 1 y 2, respectivamente.

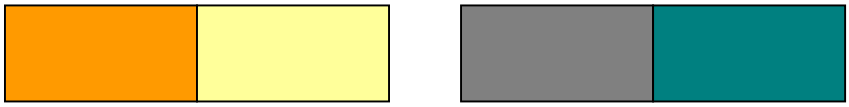


Padre 2: Vector de transporte 1y 2, respectivamente.

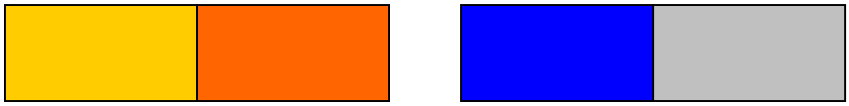


Resultados obtenidos:

Hijo 1: Vectores de transporte 1 y 2, respectivamente.



Hijo2: Vectores de transporte 1 y 2, respectivamente.



Glosario

En esta sección se van a explicar algunos conceptos que forman parte del dominio del problema que aquí se intenta resolver y que se utilizan en este documento:

CONCEPTO	DEFINICIÓN
Algoritmo genético	(AG) Método de optimización basado en la evolución biológica. Se aplica en problemas de optimización generalmente.
Base de Datos	Es una recopilación organizada de información. Hay varios tipos, nosotros utilizaremos las relacionales (se basan en el modelo de datos relacional, que almacena los datos en recopilaciones de objetos y en relaciones entre estos). Esta recopilación se guarda en un fichero que contiene los datos organizados en tablas que están relacionadas a través de éstos. Las tablas son agrupaciones de datos que definen características de los objetos de la vida real que representan. Las relaciones son asociaciones específicas entre tablas. Sobre una Base de Datos se pueden realizar acciones a través de un Sistema de Gestión de Bases de Datos Relacional (RDBMS en inglés).
DataSet	Representación de datos residente en memoria que proporciona un modelo de programación relacional coherente independientemente del origen de datos que contiene.
Flag de actividad	Variable que indica si el optimizador está o no activo. Si su valor es ‘0’, indica que el optimizador no está optimizando, mientras que si se encuentra a ‘1’, indica que el optimizador se encuentra optimizando una petición de optimización.
HTTP	(Hypertext Transfer Protocol) Es un protocolo de transmisión para el intercambio de información entre clientes y servidores en entornos basados en hipertexto/hipermedia, como es internet.
Interfaz de usuario	Indica la manera que tiene una aplicación de comunicarse con el usuario o con otras aplicaciones.
Optimizador	Aplicación que se encarga de aplicar un método de resolución basado en algoritmos genéticos para encontrar una posible solución óptima o próxima a ésta.

Petición de optimización	ver Proceso.
Planificador	Aplicación que hace de intermediario entre los clientes y los optimizadores. Se encarga de repartir las tareas entre los diferentes optimizadores que tenga a su disposición.
Problema de optimización	ver Proceso.
Proceso	Objeto complejo compuesto por una serie de datos que identifica un problema a resolver. - Una Base de Datos que con las tablas del problema. - Datos propios necesarios para su identificación.
Protocolo	Conjunto de reglas que describen el modo de utilización de alguna entidad de la vida real establecidas por convenio.
Red de Transporte	Se representa mediante un grafo en el que están interconectados los diferentes puntos (origen, intermedio y destino), que llevan asociados unas propiedades y valores de coste, por medio de arcos que también llevan asociados unos valores. Define el problema a resolver.
Servicio Web	Clase que se publica en un servidor web con soporte para ASP.NET y a cuyos métodos es posible llamar remotamente utilizando SOAP. Al terminar la ejecución del servicio web puede devolver una respuesta de algún tipo de información.
Servidor	Planificador. En ocasiones, estos dos términos se van a referir al mismo concepto, que es el Servidor. En realidad, el planificador es la parte del servidor relacionada con la asignación de un optimizador a un proceso de optimización que le llega al servidor.
SOAP	(Simple Object Access Protocol) Es un protocolo de comunicación para el intercambio de información estructurada en un entorno distribuido y descentralizado. Se utiliza para programar Servicios Web basados en tecnología XML, para definir un conjunto extensible de mensajes que se puede utilizar sobre otros protocolos. Habitualmente lo hace sobre otro protocolo de comunicación llamado HTTP.
SQL	Lenguaje utilizado para insertar y recuperar datos en la Base de Datos.
Topología	Características del grafo que representa la Red de Transporte.

UDDI	(Universal Description, Discovery and Integration) Es un servicio de descripción y localización empleado para facilitar la búsqueda de información o de servicios que presten esa información.
Utilidades	Librería desarrollada por el grupo del proyecto que implementa varias clases (opciones sobre el cliente, manejo de una base de datos...). Ésta librería la utilizarán el cliente y el servidor.
XML	(Extensible Markup Language) Es un lenguaje que se utiliza para definir lenguajes de etiquetas. Facilita la definición de interfaces para el intercambio de información entre aplicaciones.

Blibiografía

Básica

Libros

- "Profesional C# 2ª Edición". S. Robinson, K. Scott Allen, ... Ed: Wrox.Press, 2002.
- “Microsoft Visual C# .NET Aprenda ya” J. Sharp, J. Jagger. Ed: McGrawHill, 2002.
- “El lenguaje de programación C#” J.A. González Seco, 2003.
- Diferentes artículos de MSDN de distintos autores.

Enlaces

C#

- http://www.programacion.com/tutorial/csharp/
- http://www.csharphelp.com
- http://manowar.lsi.us.es/~csharp/
- http://www.csharpfriends.com/
- http://www.c-sharpcorner.com/
- http://www.dotnet.com

XML y Web Service

- http://www.it.uc3m.es/~xml
- http://www.webservicesarchitect.com/
- http://www.xml.com
- http://www.xmlsoftware.com

Algoritmos genéticos

- http://wwwdi.ujaen.es/~mlucena
- http://kal-el.ugr.es/~jmerelo

Complementaria

Enlaces

- http://www.elguille.info/NET/indice.asp
- http://www.programacion.com/
- http://msdn.microsoft.com/
- http://www.microsoft.com/spanish/msdn/botica.asp